ESD-TR-90-019

Technical Report
886

MUSE — A Systolic Array for Adaptive Nulling with 64 Degrees of Freedom, Using Givens Transformations and Wafer Scale Integration

C.M. Rader
D.L. Allen
D.B. Glasco
C.E. Woodward

18 May 1990

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

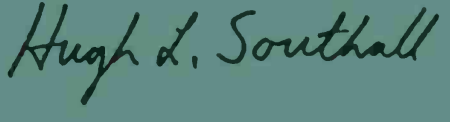
**Prepared for the Department of the Air Force
under Contract F19628-90-C-0002.**

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Department of the Air Force under Contract F19628-90-C-0002.

The ESD Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



Hugh L. Southall, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

MUSE — A SYSTOLIC ARRAY FOR ADAPTIVE NULLING WITH 64 DEGREES OF FREEDOM, USING GIVENS TRANSFORMATIONS AND WAFER SCALE INTEGRATION

C.M. RADER
D.B. GLASCO
Group 27

D.L. ALLEN
C.E. WOODWARD
Group 23

TECHNICAL REPORT 886

18 MAY 1990

LEXINGTON MASSACHUSETTS

ABSTRACT

This report describes an architecture for a highly parallel system of computational processors specialized for real-time adaptive antenna nulling computations with many degrees of freedom, called MUSE, and a specific realization of MUSE for 64 degrees of freedom. Each processor uses the CORDIC algorithm and has been designed as a single integrated circuit. Ninety-six such processors working together can update the 64-element nulling weights based on 300 new observations in only 6.7 ms. This is equivalent to 2.88 Giga-ops for a conventional processor. The computations are accurate enough to support 50 dB of signal-to-noise improvement in a sidelobe canceller. The connectivity between processors is quite simple and permits MUSE to be realized on a single large wafer, using restructurable VLSI. The complete design of such a wafer is described.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

When an array of N antenna elements is subject to undesired interference, such as co-channel interference or jamming plus the thermal noise in each of the N receivers, the interference power can be reduced, relative to the power in some desired signal, by forming a suitable weighted sum of the waveforms observed on all the antenna elements as the system output. This is called *nulling*. In many instances the choice of suitable weights must be made adaptively with time. The optimum choice of weights, in the sense that the signal-to-interference ratio observed in the system output is maximized, is the solution to a well studied least-squares problem. The number of arithmetic steps required to solve this least-squares problem, almost independent of the algorithm chosen, is proportional to the cube of the number of antenna elements. Furthermore, in the presence of interference that may change with time, it is necessary to perform the adaptive weight determination repetitively and in real time.

When the antenna is on board a moving spacecraft, the combination of a real-time requirement driven by the satellite motion and the cubic dependence of the computational cost of adaptive weight determination gives a limit to the number of antenna elements that may practically be nulled. A previous study of this computational cost [1] set this limit at about $N = 26$, based on an assumed conventional digital signal processing architecture. Such a limitation is in no sense absolute, since it depends on what resources we are willing to allocate to a nulling processor. However, because the difficulty of the nulling computation grows as N^3, simply using more resources is not an efficient way to handle a large number of antennas.

In this technical report, a specialized adaptive nulling processor called MUSE,[1] which is capable of determining the weights for $N = 64$ antenna elements, is described. Due to its novel architecture, it may be realized compactly using restructurable wafer scale integration; it would then be the size of the square inscribed in a 5-in-diam. circle. MUSE is not only substantially smaller and lighter than a conventional processor, but it also will use substantially less power. Although the MUSE processor is specialized for $N = 64$ antennas, the design concept of MUSE can be applied to design a similar processor for an antenna array with either more or fewer elements.

In the next section there is a brief explanation of the mathematics of adaptive nulling and the method of voltage domain computation of a Cholesky factor using Givens transformations. In section 3, there is an explanation of CORDIC rotations and a demonstration of how they are suitable for realizing the Givens transformations needed in updating a Cholesky factor. In section 4, the idea of a "naive" systolic array of computing elements, each of which is composed of three CORDIC rotation cells operating together, is developed. So connected, an array of computing elements is capable of sharing the work of updating a Cholesky factor, which is the largest part of the computational task. However, it is relatively inefficient. In section 5, we show how to modify the systolic array to make it 100 percent efficient for the Cholesky update task, using a technique which is called latency-controlled interleaving. In section 6, the parameter choices for the CORDIC cell, influenced by the specific system goals, are explained. A simulation used to motivate the choice of

[1] MUSE is an acronym which stands for Matrix Update Systolic Experiment.

some of these parameters is described. In section 7, we describe some of the technology issues of restructurable VLSI and show how the systolic array is suited to implementation in this technology.

In section 8, the problem of solving the linear equations for the optimum weights, once a Cholesky factor is found, is discussed. CORDIC cells can be used for this computation in a manner that is quite closely related to the way in which they are used to update a Cholesky factor. A modification is described of the timing of the systolic array so that it may be used for both the Cholesky update and the solution of the linear equations.

In section 9, a detailed description is provided of a cell that performs the CORDIC function and all related necessary functions for building up a MUSE processor on a wafer from 96 identical cells of that type. In section 10, some specifics are given of the topology of such a wafer and of the method by which the cells are connected together to form a working system. In section 11, the architecture of a test and demonstration system for demonstrating the performance of such an integrated wafer is described.

2. VOLTAGE DOMAIN NULLING WITH GIVENS TRANSFORMATIONS

Figure 2-1 is a cartoon of a typical adaptive nulling system. The N antenna elements each produce a waveform that is assumed to be down-converted, sampled, and digitized. At the n-th sampling instant a complex number x_{in} comes from the ith antenna. All the antennas are assumed to be sampled at the same instant, so we may collect together the samples from the same sampling instant into a column vector $\underline{X}(n)$. These vector samples are passed to the right, where the nulling takes place, and some samples are also passed down to where the adaptation takes place.

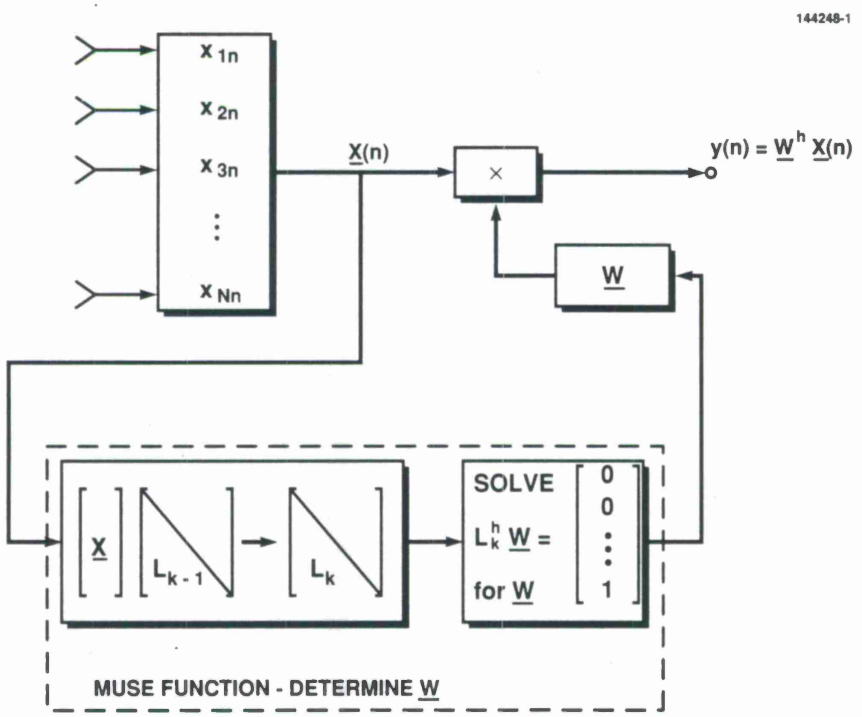


Figure 2-1. Cartoon of typical adaptive nulling system.

The adapted weights are also collected into a column vector, which will be labeled as $\underline{W}$. Although $\underline{W}$ will be time-varying, it will not be recomputed for every sampling interval. A *block* is defined as the interval within which $\underline{W}$ will be held constant. The nulled output, $y(n)$, a scalar, is formed by using weights $\underline{W}$ on the vector $\underline{X}(n)$. We can use the vector notation

$$y(n) = \underline{W}^h \underline{X}(n),$$

where $()^h$ means Hermitian transpose. This notation implies that we actually multiply the elements of $\underline{X}(n)$ by the complex conjugates of the components of $\underline{W}$.

In Figure 2-1 the adaptation process involves samples $\underline{X}(n)$ and also a lower triangular matrix, L_k, which is regularly updated and then used to determine $\underline{W}$. The process of adaptation will now be explained.

Within each block, the optimization of $\underline{W}$ is with respect to the statistics of interference appropriate to that block. The statistics that matter are the correlations of interference observed from one antenna element to another, arranged into a matrix R. A mathematical description of the nulling system's performance will refer to this correlation matrix of the interference, which is generally not known in practice since it is a statistical expectation.

$$R = \mathcal{E}\left(\underline{J}(n)\underline{J}^h(n)\right)$$

where $\underline{J}(n)$ is the interference component of $\underline{X}(n)$. R is never perfectly known for two reasons:

- One sees the interference and the desired signal at the same time as an inseparable sum.
- One has neither the time to gather enough samples nor the computational resources to use all the samples available to estimate R perfectly.

We are going to make the simplifying assumption that although the observed antenna signals $\underline{X}(n)$ contain both desired signals and interference, the desired signals are very much weaker than the interference and may be neglected in forming the correlation matrix. This simplifying assumption is reasonable because a radar designer always tries to get by with the least possible transmitter power compatible with detecting a target after signal processing. Since signal processing enhances the target return relative to interference, it follows that the designer will have chosen the transmitter power small enough that the target return before any signal processing will be below the noise. Therefore, the first limitation is not a problem[1] and only the second is serious. A limited amount of data must be used to obtain an estimate of R for each block and then, since R is changing, another limited set of data may be used each time R is updated, and so on.

Let us therefore collect together all the samples of $\underline{X}(n)$ that we intend to use in determination of "optimum" weights for a given block and refer to this collection of data as X. There is no implication that all the samples used come from within the given block – we actually envision using samples from within a sliding window. If the number of samples used is M and each sample is an N element vector, it is natural to arrange the M vectors into a matrix of M columns and N rows. The estimate of the correlation matrix based on this matrix of raw data is

$$\hat{R} = \frac{1}{M} X X^h.$$

[1] This argument assumes that only small targets are to be detected. If a mix of small and large targets are sought, it may not be appropriate to neglect the largest targets when estimating interference statistics.

Since we want to optimize a signal-to-noise (S/N) ratio, we need to characterize the signal. Let $\underline{S}$ be a known constant vector related to the desired signal as follows: if all interference were absent and only signal present, we would expect to observe, in $\underline{X}(n)$, some scalar time function times the vector $\underline{S}$.

Then the optimum choice of $\underline{W}$ is known from the literature [2] to satisfy the equation

$$R\underline{W} = \underline{S}.$$

This equation is of great importance to us for two reasons. First, the only information needed from the block X to determine the optimum weight vector is that information which could be used to estimate R. Hence, if we intend to estimate R using $\hat{R} = \frac{1}{M}XX^h$, we can be comfortable in transforming X in any way so long as that transformation leaves XX^h unchanged. Second, the equation tells us that we can find $\underline{W}$ from R by solving linear equations and hence it suggests that we might accept as a pretty good estimate [3] of $\underline{W}$ the solution to the same equations using $\hat{R}$,

$$\hat{R}\underline{W} = \underline{S}.$$

How can X be transformed while leaving XX^h unchanged? Suppose we postmultiply X by a unitary matrix Q:

$$\hat{X} = XQ.$$

By definition a unitary matrix has the property

$$QQ^h = I$$

so we may insert QQ^h between X and X^h in the equation for $\hat{R}$

$$\hat{R} = XX^h = XQQ^hX^h = \hat{X}\hat{X}^h.$$

That means X can be transformed into $\hat{X}$ without changing the correlation matrix that is estimated from it. Furthermore, this transformation may be repeated again with another unitary matrix, and still again, as often as liked, until the final transformed version of X has a convenient form.

The strategy will be to make successive choices of Q that cause selected elements in the matrix XQ to become zeroes, but such that elements of X that have already been zeroed by previous transformations are not changed. A series of such transformations can confine the nonzero elements of the $N \times M$ matrix X to an $N \times N$ lower-triangular submatrix, which is called L, on the left, e.g.,

$$\begin{bmatrix} x & x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & x & x \end{bmatrix} \Rightarrow \left[\begin{array}{cccc|c} l & & & & \\ l & l & & & \\ l & l & l & & \bigcirc \\ l & l & l & l & \end{array}\right] \equiv [L \mid 0].$$

The matrices L_{k-1} and L_k in Figure 2-1 are instances of L.

Having made this sequence of transformations we can write

$$\frac{1}{M}XX^h = \hat{R} = \frac{1}{M}LL^h,$$

and then the linear equations we must solve for $\underline{W}$ take the simpler form

$$LL^h\underline{W} = M\underline{S}.$$

The known constant M may be absorbed into the steering vector $\underline{S}$ and is hereafter omitted. (Note that since $\underline{W}$ is the solution to a set of linear equations, if $\underline{S}$ is scaled by any constant, we scale the solution $\underline{W}$ by the same constant. However, the signal-to-interference ratio in $\underline{W}^h\underline{X}(n)$ is unchanged by multiplying $\underline{W}$ by a scale factor.)

We can solve $LL^h\underline{W} = \underline{S}$ in two easy steps. First define an intermediate vector variable $\underline{Y}$ defined by

$$L^h\underline{W} = \underline{Y}, \tag{2.1}$$

so that $\underline{Y}$ is the solution to

$$L\underline{Y} = \underline{S}. \tag{2.2}$$

We can first solve Equation (2.2) for $\underline{Y}$ and then, having found $\underline{Y}$, we are in position to solve Equation (2.1) for $\underline{W}$. Both these equations are easier to solve than general matrix/vector equations because the matrices involved are triangular. If we had to solve a matrix/vector equation involving a general $N \times N$ square matrix, we would expect to require a number of arithmetic steps proportional to N^3. However, when the matrix is triangular, the straightforward method of solution requires a number of arithmetic steps proportional to only N^2.

To illustrate this, consider a 4×4 special case of Equation (2.2) and Equation (2.1):

$$l_{11}y_1 = s_1 \tag{2.3}$$

$$l_{21}y_1 + l_{22}y_2 = s_2 \tag{2.4}$$

$$l_{31}y_1 + l_{32}y_2 + l_{33}y_3 = s_3 \tag{2.5}$$

$$l_{41}y_1 + l_{42}y_2 + l_{43}y_3 + l_{44}y_4 = s_4 \tag{2.6}$$

$$l^*_{44}w_4 = y_4 \tag{2.7}$$

$$l^*_{43}w_4 + l^*_{33}w_3 = y_3 \tag{2.8}$$

$$l^*_{42}w_4 + l^*_{32}w_3 + l^*_{22}w_2 = y_2 \tag{2.9}$$

$$l^*_{41}w_4 + l^*_{31}w_3 + l^*_{21}w_2 + l^*_{11}w_1 = y_1 \tag{2.10}$$

From Equation (2.3) we immediately get

$$y_1 = \frac{s_1}{l_{11}},$$

and the now known value of y_1 maybe used in the next three equations, giving:

$$\begin{aligned} l_{22}y_2 &= s_2 - l_{21}y_1 \equiv s_2' \\ l_{32}y_2 + l_{33}y_3 &= s_3 - l_{31}y_1 \equiv s_3' \\ l_{42}y_2 + l_{43}y_3 + l_{44}y_4 &= s_4 - l_{41}y_1 \equiv s_4' \end{aligned}$$

This shows that we solve for one unknown in an $N \times N$ triangular system and also reduce the problem to an $(N-1) \times (N-1)$ triangular system at a cost of $N-1$ multiply-and-add operations and one division.

Now we immediately find $y_2 = s_2'/l_{22}$ and reduce to a 2×2 triangular system in an analogous manner. Taking this process to the end, we will need to use N divisions and $(N-1)+(N-2)+\ldots+1$ multiply-and-add operations. The series sums to $\frac{1}{2}N(N-1)$, which approaches $\frac{1}{2}N^2$ for large N.

After using Equations (2.3) through (2.6) to find y_1, y_2, y_3, y_4, these can be used as the known right-hand side in Equations (2.7) through (2.10). Thus

$$w_4 = \frac{y_4}{l_{44}^*},$$

and the now known value of w_4 may be used in Equations (2.8), (2.9), and (2.10).

$$\begin{aligned} l_{33}^* w_3 &= y_3 - l_{43}^* w_4 \equiv y_3' \\ l_{32}^* w_3 + l_{22}^* w_3 &= y_2 - l_{42}^* w_4 \equiv y_2' \\ l_{31}^* w_3 + l_{21}^* w_3 + l_{11}^* w_1 &= y_1 - l_{41}^* w_4 \equiv y_1' \end{aligned}$$

Now we have

$$w_3 = \frac{y_3'}{l_{33}^*},$$

and we may continue in an analogous manner. The operation count for solving the second triangular set of equations is identical to that for the first set.[2]

The above approach to adaptive nulling may be broken into several phases:

Obtain typical interference: Form a rectangular matrix X whose columns are the (vector) samples of interference observed at the antenna elements.

[2] If the antenna array is designed such that the appropriate steering vector is $\underline{S} = [0, 0, \ldots, 0, 1]^t$, the first triangular set of equations becomes trivial and only the second set needs to be solved. This kind of adaptive antenna array is known as a sidelobe canceller.

Triangularize: Use orthogonal transformations to compute a triangular matrix L from X.

Back-substitute: Solve the set $L\underline{Y} = \underline{S}$ for the intermediate variables $\underline{Y}$.

Back-substitute: Solve the set $L^h\underline{W} = \underline{Y}$ for the desired weight vector $\underline{W}$.

This may be contrasted with a mathematically equivalent approach in which we compute

$$\hat{R} = XX^h$$

followed by a procedure called Cholesky factorization for determining L from $\hat{R}$ and then carry out the two back-substitutions. The latter approach, while valid, is not numerically robust. In the matrix $\hat{R}$, the largest and smallest eigenvalues are usually determined, respectively, by jammer power and thermal noise power. This ratio, if very large, determines the minimum word-length needed in numerical operations to solve linear equations involving $\hat{R}$. This word-length limitation is fundamental and does not depend on the algorithm used to solve the linear equations. But if L is determined directly from the raw data X, without ever computing $\hat{R}$, the minimum word-length requirement for solving linear equations involving L or L^h is about half that needed with $\hat{R}$.

Any nulling algorithm that depends on explicit computation of $\hat{R}$ from X is called a *power domain* algorithm. By contrast, an algorithm that determines L by transforming the data set X is called a *voltage domain* algorithm. Because it is most economical to design and construct hardware with short word-lengths, voltage domain algorithms are preferred.

Now let us give some attention to the treatment of successive blocks. For this we must recognize that both X and L will be different in different blocks. Let the raw data and the Cholesky matrix in the kth block be X_k and L_k. But how is X_k related to X_{k-1}?

If the statistics of interference were expected to remain constant with time, then we would expect to estimate those statistics better and better by using more and more raw data in the successive blocks. We would append the observations that became available during the kth block, X_{new}, onto the matrix of all the observations available before, X_{k-1}:

$$X_k = [X_{k-1} \mid X_{new}]$$

so that

$$\hat{R}_k = \hat{R}_{k-1} + X_{new}X_{new}^h$$

and

$$L_kL_k^h = L_{k-1}L_{k-1}^h + X_{new}X_{new}^h$$

$$L_kL_k^h = [L_{k-1} \mid X_{new}][L_{k-1} \mid X_{new}]^h.$$

This suggests an efficient way to update L by use of unitary transformations. If Q is a unitary transformation, then

$$L_k L_k^h = [L_{k-1} \mid X_{new}] Q \, [[L_{k-1} \mid X_{new}] Q]^h .$$

We choose Q so as to force zeroes into the positions occupied by X_{new}, thus

$$\left[\begin{array}{cccc|cccccc} l & & & & n & n & n & n & n & n \\ l & l & & & n & n & n & n & n & n \\ l & l & l & & n & n & n & n & n & n \\ l & l & l & l & n & n & n & n & n & n \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} l & & & & \\ l & l & & & \\ l & l & l & & \bigcirc \\ l & l & l & l & \end{array}\right] .$$

There are two advantages to updating L_k using L_{k-1}. The first advantage is that use is made of prior work, saving computation. Second, and more important, is that X_k involves more and more data, hence more and more storage as k increases. The storage of L_k requires only N^2 real numbers.

Since the interference statistics are expected to change with time, we would not want to use the scheme described above without modification. But interference statistics change slowly in comparison with the duration of a block (or else we would not be able to hold the weight vector constant over that duration); therefore we would like to make some use of previous blocks. An often used compromise is to weight the data by an exponentially decaying window. An easy way to accomplish this, using a "forgetting factor" of α, is to use αL_{k-1} in place of L_{k-1} in the previous algorithm for updating L_k.

Since L can be updated recursively, we can now update L *for each new vector sample as it becomes available* [3] and preserve a *snapshot* of $L = L_k$ after the update that represents the end of the kth block. This makes the update rule very simple. Let $\underline{X}$ be the new vector sample and let L_{old} be the Cholesky matrix for all previous data. Let α be the forgetting factor.[4] A unitary matrix Q of size $(N+1) \times (N+1)$ is used in the update:

$$[\alpha L_{old} | \underline{X}] \, Q \Rightarrow [L_{new} | \underline{0}] .$$

Now the mechanization of the unitary transformation can be discussed. As previously explained, we can postmultiply by a sequence of simple unitary transformations to accomplish the same effect as a single more elaborate transformation. Since the purpose is to introduce zeroes into

[3] In an earlier study by Steinhardt [5], we examined the approach of updating L for many new samples at once. The appropriate unitary transformation for multiple sample updates is a Householder transformation. Householder transformations require less arithmetic than the approach taken in this report, but they have not yet been shown to have a simple realization in highly parallel systolic hardware.

[4] α is now chosen based on updating L on a per-sample basis rather than on a per-block basis.

the tacked-on column $\underline{X}$ of the matrix $[\alpha L_{old}|\underline{X}]$, we will use unitary transformations which each zero out one additional element.

Two types of unitary transformations are used. One type makes the leading nonzero element of the tacked-on vector, which is probably a complex number, become a real number. The unitary matrix Q_θ which accomplishes this is of the form

$$Q_\theta = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & e^{j\theta} \end{bmatrix};$$

that is to say, Q_θ is an $(N+1)\times(N+1)$ identity matrix with the lower right diagonal element replaced by $e^{j\theta}$. Postmultiplying $[\alpha L_{old}|\underline{X}]$ by Q_θ leaves most of the matrix unchanged but multiplies the last column (the tacked-on vector) by $e^{j\theta}$. We must choose θ such that multiplying the leading element of the tacked-on vector by $e^{j\theta}$ makes the product real.

The other type of unitary transformation used is called a Givens transformation and is denoted by Q_ϕ. The form of the matrix Q_ϕ is derived from an $(N+1)\times(N+1)$ identity matrix by changing four elements. The mth and $(N+1)$-st diagonal elements become $\cos\phi$. The element in row $(N+1)$ and column m becomes $\sin\phi$. The element in column $(N+1)$ and row m becomes $-\sin\phi$. This is illustrated here for $m = 2$:

$$Q_\phi = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \cos\phi & 0 & \dots & 0 & -\sin\phi \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & \sin\phi & 0 & \dots & 0 & \cos\phi \end{bmatrix}.$$

When we postmultiply by Q_ϕ only the mth column and the tacked-on $(N+1)$-th column are affected. Suppose that the elements in the same row in those two columns are l and x. After the transformation

$$\begin{aligned} l' &\Leftarrow l\cos\phi + x\sin\phi, \text{and} \\ x' &\Leftarrow x\cos\phi - l\sin\phi. \end{aligned}$$

If l and x are real, this may be viewed as a geometric rotation through the angle ϕ in the plane defined by l and x axes.

By an alternation of postmultiplications by unitary matrices of these two types, the entire tacked-on column can be zeroed out. This progression is illustrated in Figure 2-2. The symbolism

in this figure uses ordinary letters to indicate complex quantities and letters with an overbar, $\bar{a}$, to indicate quantities that must be real. In addition, if a column in a matrix on the left is in boldface, that column is affected by the transformation. Other columns are not affected. An example with $N = 4$ is illustrated.

When the mechanics of the two types of unitary transformation is compared, there is a surprising commonality. For the Q_θ we look at the leading nonzero element of the tacked-on column and determine the "angle" θ that will make this leading element real. Because the column is composed of complex numbers, the determination of θ depends on two real numbers that are, respectively, the real part and the imaginary part of the leading element x_i:

$$\theta = \arctan \frac{-Im(x_i)}{Re(x_i)}.$$

When $\cos\theta$ and $\sin\theta$ are available, the transformation Q_θ is applied to the remainder of the tacked-on column, which we may consider as two columns of real numbers that are its real and imaginary parts. In terms of the two columns, the ith Q_θ transformation takes the form:

$$\begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ Re(x_i) & Im(x_i) \\ Re(x_{i+1}) & Im(x_{i+1}) \\ \vdots & \vdots \\ Re(x_N) & Im(x_N) \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

On the other hand, for the ith Givens transformation Q_ϕ, the determination of the "angle" ϕ is made on the basis of the leading elements of two columns. $l_{i,i}$ is the leading element of column i and x_i is the leading element of the tacked-on column. The leader of the tacked-on column is real as a result of the preceding Q_θ transformation. Now assume that $l_{i,i}$ is real – therefore by mathematical induction it will remain real because all the operations involved in updating it will produce real numbers. ϕ is determined by

$$\phi = \arctan \frac{-x_i}{l_{i,i}},$$

which is exactly analogous to the determination of θ for the Q_θ transformation. Then, when $\cos\phi$ and $\sin\phi$ are available, the Givens transformation is applied to the only two columns affected by it. The Givens transformation may be written in terms of these two columns:

$$\begin{bmatrix} \bar{l} & & & & \mathbf{x} \\ l & \bar{l} & & & \mathbf{x} \\ l & l & \bar{l} & & \mathbf{x} \\ l & l & l & \bar{l} & \mathbf{x} \end{bmatrix} Q_\theta \Rightarrow \begin{bmatrix} \bar{l} & & & & \bar{x} \\ l & \bar{l} & & & x \\ l & l & \bar{l} & & x \\ l & l & l & \bar{l} & x \end{bmatrix}$$

$$\begin{bmatrix} \bar{\mathbf{l}} & & & & \bar{\mathbf{x}} \\ \mathbf{l} & \bar{l} & & & \mathbf{x} \\ \mathbf{l} & l & \bar{l} & & \mathbf{x} \\ \mathbf{l} & l & l & \bar{l} & \mathbf{x} \end{bmatrix} Q_\phi \Rightarrow \begin{bmatrix} \bar{l} & & & & 0 \\ l & \bar{l} & & & x \\ l & l & \bar{l} & & x \\ l & l & l & \bar{l} & x \end{bmatrix}$$

$$\begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \mathbf{x} \\ l & l & \bar{l} & & \mathbf{x} \\ l & l & l & \bar{l} & \mathbf{x} \end{bmatrix} Q_\theta \Rightarrow \begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \bar{x} \\ l & l & \bar{l} & & x \\ l & l & l & \bar{l} & x \end{bmatrix}$$

$$\begin{bmatrix} \bar{l} & & & & \\ l & \bar{\mathbf{l}} & & & \bar{\mathbf{x}} \\ l & \mathbf{l} & \bar{l} & & \mathbf{x} \\ l & \mathbf{l} & l & \bar{l} & \mathbf{x} \end{bmatrix} Q_\phi \Rightarrow \begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & 0 \\ l & l & \bar{l} & & x \\ l & l & l & \bar{l} & x \end{bmatrix}$$

$$\begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{l} & & \mathbf{x} \\ l & l & l & \bar{l} & \mathbf{x} \end{bmatrix} Q_\theta \Rightarrow \begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{l} & & \bar{x} \\ l & l & l & \bar{l} & x \end{bmatrix}$$

$$\begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{\mathbf{l}} & & \bar{\mathbf{x}} \\ l & l & \mathbf{l} & \bar{l} & \mathbf{x} \end{bmatrix} Q_\phi \Rightarrow \begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{l} & & 0 \\ l & l & l & \bar{l} & x \end{bmatrix}$$

$$\begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{l} & & \\ l & l & l & \bar{l} & \bar{\mathbf{x}} \end{bmatrix} Q_\theta \Rightarrow \begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{l} & & \\ l & l & l & \bar{l} & \bar{x} \end{bmatrix}$$

$$\begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{l} & & \\ l & l & l & \bar{\mathbf{l}} & \bar{\mathbf{x}} \end{bmatrix} Q_\phi \Rightarrow \begin{bmatrix} \bar{l} & & & & \\ l & \bar{l} & & & \\ l & l & \bar{l} & & \\ l & l & l & \bar{l} & 0 \end{bmatrix}$$

Figure 2-2. Eight successive unitary transformations which zero out the four element tacked-on vector.

$$\begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ l_{i,i} & x_i \\ l_{i+1,i} & x_{i+1} \\ \vdots & \vdots \\ l_{N,i} & x_N \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}.$$

This shows that the mechanization of the Givens transformation is almost exactly the same as the mechanization of the Q_θ transformation. There is one difference. After dealing with the leaders, the remainder of the Givens transformation is applied to two columns of complex numbers, whereas the Q_θ transformation is applied to two columns of real numbers. However, we can consider the Givens transformation applying separately to the real parts of its two affected columns and to the imaginary parts of its two affected columns. In these terms a Givens transformation is identical to two Q_θ transformations. The unitary transformations needed to zero out a tacked-on column appended to a (triangular) Cholesky matrix are all identical and can be implemented using identical hardware. In the subsequent sections a design for such hardware in detail will be described.

It is instructive to estimate the total workload involved in using the algorithm we have described to zero out a single 64-element vector. Each element in the Cholesky matrix is modified using three rotations. To carry out each rotation using a conventional computer architecture would appear to require two load instructions, four multiplications, an addition, a subtraction, and two stores, about ten instructions per rotation.[5] Therefore $3(N+(N-1)+(N-2)+\ldots+1) = \frac{3}{2}N(N+1)=$ 6240 rotations, equivalent to about 62,000 instructions, are involved per new sample.

Several authors have proposed parallel computation to compute the update of a Cholesky matrix. The best known architecture is a systolic array of computing elements arranged in a triangular mesh [4]. For a given number of degrees of freedom, N, such a triangular array uses $N(N-1)/2$ processors. This limits practical application of the triangular array architecture to very small N.

[5] The steps involved to rotate the leaders are different and more involved, but we are counting them as if they were the same.

3. CORDIC REALIZATION OF GIVENS TRANSFORMATION

In this section, an algorithm is described for coordinate rotation that is well suited for digital realization, known as CORDIC. The acronym stands for coordinate rotation digital computation. The basic idea was first published in the 1950s by Volder [6].

Suppose a vector points from the origin to a point whose coordinates are (x, y), and that this vector is to be rotated to a new point with coordinates (x', y'), such that the angle between the new and old points is ξ. We have

$$\begin{aligned} x' &= (\cos\xi)(x - y\tan\xi), \text{and} \\ y' &= (\cos\xi)(y + x\tan\xi). \end{aligned}$$

This involves four multiplications. However, there are many special angles for which some of the multiplications would simplify to shifts. We will concentrate on the special angles ξ_ν for which

$$\tan\xi_\nu = \pm 2^{-\nu}.$$

Then the multiplications by $\tan\xi_\nu$ become right-shifts by ν bit positions. For fixed ν, the two special angles are of the same magnitude but opposite sign and therefore they have the same cosine.

The first step of the CORDIC algorithm is to realize any arbitrary angle ξ by a sequence of rotations either forward or backward by ξ_ν, $\nu = 0, 1, \ldots, \infty$. Let $\rho_\nu = \pm 1$ determine whether a particular "minirotation" is forward or backward. Thus,

$$\xi = \sum_{\nu=0}^{\infty} \rho_\nu \xi_\nu,$$

and the rotation of (x, y) through the angle ξ is accomplished by the sequence of steps in Figure 3-1.

The key step in development of the CORDIC approach is to recognize that the multiplications by $\cos\xi_\nu$ may all be collected together into a single constant

$$K = \prod_{\nu=0}^{\infty} \cos\xi_\nu$$

that is independent of the overall angle ξ by which we rotate. Thus Figure 3-1 can be revised to the form of Figure 3-2.

The CORDIC algorithm is composed of "stages"—most of the stages perform minirotations and a "last" stage performs a gain correction. For any angle, all the minirotations are employed, but each is employed in either a clockwise or counterclockwise direction. Although an infinite number of minirotations are called for in an exact realization of the CORDIC algorithm, it is practical to use a finite number of stages, in the same sense that it is practical to make a binary adder or

$$\begin{pmatrix} x \\ y \end{pmatrix} \Leftarrow \cos\xi_0 \begin{pmatrix} x - 2^{-0}\rho_0 y \\ y + 2^{-0}\rho_0 x \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} \Leftarrow \cos\xi_1 \begin{pmatrix} x - 2^{-1}\rho_1 y \\ y + 2^{-1}\rho_1 x \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} \Leftarrow \cos\xi_2 \begin{pmatrix} x - 2^{-2}\rho_2 y \\ y + 2^{-2}\rho_2 x \end{pmatrix}$$

$$\vdots$$

Figure 3-1. Achieving rotation through ξ using minirotations through the special angles $\pm\xi_\nu$.

$$\begin{pmatrix} x \\ y \end{pmatrix} \Leftarrow \begin{pmatrix} x \\ y \end{pmatrix} + 2^0\rho_0 \begin{pmatrix} -y \\ x \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} \Leftarrow \begin{pmatrix} x \\ y \end{pmatrix} + 2^{-1}\rho_1 \begin{pmatrix} -y \\ x \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} \Leftarrow \begin{pmatrix} x \\ y \end{pmatrix} + 2^{-2}\rho_2 \begin{pmatrix} -y \\ x \end{pmatrix}$$

$$\vdots$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} \leftarrow K \begin{pmatrix} x \\ y \end{pmatrix}$$

Figure 3-2. Rotation through ξ using minirotations through the special angles $\pm\xi_\nu$ using one correction gain.

multiplier with a finite number of stages— the higher numbered stages contribute very little to the accuracy of the angle specification. The correction stage is a multiplication by the fixed quantity K, not a general-purpose multiplier. The exact value of K depends on how many CORDIC stages are used. As Table 3-1 shows, $K(\nu_{max})$ converges quite quickly. (In the MUSE application, the fixed multiplication by K, designed into each CORDIC circuit, is combined with another multiplication called the forgetting factor.)

TABLE 3-1.

Correction Constant Required when the Last Stage Rotates by $\arctan 2^{-\nu_{max}}$

ν_{max}	K	ν_{max}	K
0	0.707106781	8	0.607254479
1	0.632455532	9	0.607253321
2	0.613571991	10	0.607253031
3	0.608833912	11	0.607252959
4	0.607648256	12	0.607252941
5	0.607351770	13	0.607252936
6	0.607277644	14	0.607252935
7	0.607259112	15	0.607252935

A CORDIC method therefore achieves rotation without using any of the trigonometric functions and without explicit multiplications. If the angle by which we wish to rotate the pair (x, y) is known in advance, we can determine the set of controls

$$(\rho_i, i = 0, \ldots, \nu_{max})$$

that are each represented by a single bit. This may be thought of as representing the angle ξ_i using digits ρ_i in an unconventional number system, a number system different from binary, decimal, or any of the radix systems that are commonly used. Each ρ_i can be stored in a flip-flop in the stage whose direction of rotation it is to control.

However, in the Givens transformation application the angle of rotation is not known in advance. A pair (x, y) is given and we must rotate that pair through the angle such that the resulting rotated pair takes the form $(x', 0)$. This operation is called *vectoring*. Then we must rotate some number of other pairs through the same angle. There is no need to know what the angle is, so long as we are able to rotate by that angle. Therefore what we really need for vectoring is an algorithm to determine the CORDIC controls ρ_i. A major advantage of the CORDIC algorithm is that the same circuit that is used for rotating may be used for vectoring.

Consider just the first CORDIC stage, for which the special angle is either $45°$ or $-45°$. Since the purpose is to rotate the input (x, y) toward the x axis, if y is above the axis then we should rotate "down," and if y is below the axis then we should rotate "up." Therefore

$$\rho_0 = sgn(x)sgn(y).$$

Once ρ_0 is determined we compute the effect of the first stage on x and y and pass (x, y) as modified to the second stage. Here again the rule is to rotate down if y is above the axis and up if y is below the axis.

$$\rho_1 = sgn(x)sgn(y).$$

In this way, the determination of the controls is quite simple.

$$\rho_i = sgn(x)sgn(y), \quad i = 0, \ldots, \nu_{max}.$$

These controls, once determined, are saved in the flip-flops of the specialized stages and are used to control those stages for the succeeding (x, y) pairs that are to be rotated through the same angle.

In Figure 3-3 the concept of a CORDIC circuit made up of independent stages is shown.[1] The inputs (x, y) are modified by minirotations as they proceed from stage to stage. This circuit has the virtue of natural pipelining. If registers are placed between the stages where there are dashed lines in Figure 3-3, then a new rotation problem, involving a new pair (x, y), can be started by

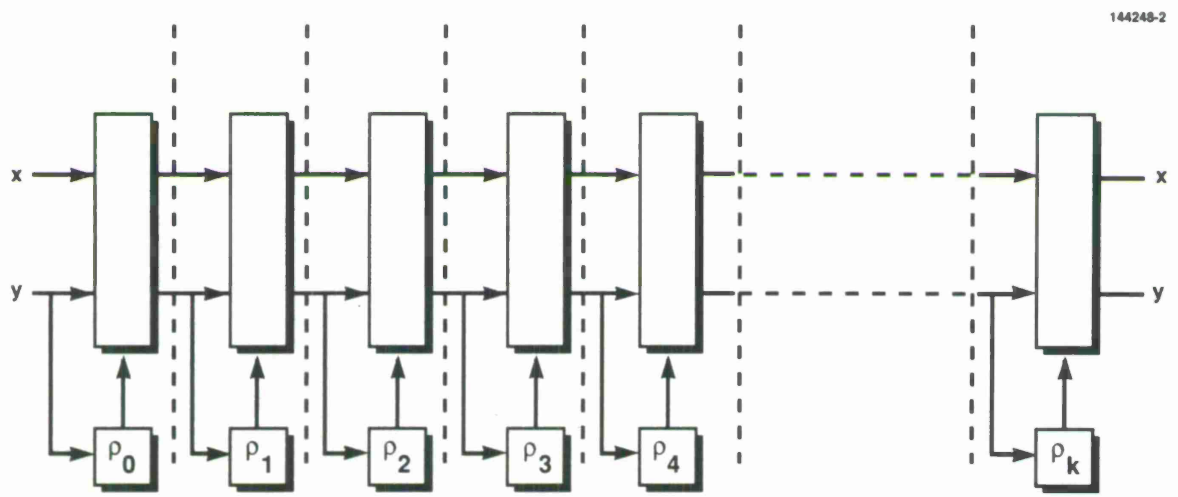


Figure 3-3. Pipelined CORDIC *circuit.*

the circuit as soon as the preceding pair has been latched at the output of the first stage; then yet another rotation may be started when the first two pairs have been latched at the output of the second and first stages, respectively, etc. Note that rotation may follow vectoring in this pipelined

[1] In Figure 3-3 the controls ρ_i are shown as if they depended only on the $sgn(y)$. The simplified figure is meant only to illustrate the pipelining aspect of CORDIC circuits. In fact the controls depend on $sgn(y)sgn(x)$.

fashion. Only an addition and a subtraction need to be performed in each stage. These operations can be carried out very rapidly in digital logic. Therefore the rate at which a CORDIC circuit can begin new rotation problems is quite high. By contrast the time required to complete any given rotation is proportional to the number of CORDIC stages provided.

4. CORDICS IN SYSTOLIC ARRAY FOR UPDATING A CHOLESKY MATRIX

The pipelined CORDIC circuit is ideally used as one of a large number of computing elements operating in parallel in a systolic array. The term "systolic" originated in human biology to describe the system of circulation of the blood through blood vessels and various organs. In a systolic computing system the data is "pumped" from computing organ to computing organ and processed as it moves. Ideally in a systolic system, data should be processed as soon as it becomes available, so it must be available at the right place at the right time. This means that the algorithmic and the architectural considerations are very closely intertwined.

4.1 Supercell

In this section we will show how a large number of the pipelined CORDIC circuits can be used in a parallel systolic pipelined system to update a Cholesky matrix. One CORDIC circuit will be assigned to perform each of the N Q_θ transformations, and two CORDIC circuits will be assigned to perform each of the N Q_ϕ transformations. Therefore three CORDIC circuits are needed to perform all the operations associated with updating one of the N columns of the Cholesky matrix and $3N$ CORDIC circuits are needed in all.

A common data format will be used throughout. Data that are vectors of complex numbers are presented to circuitry sequentially, and the time during which one complex word enters a CORDIC circuit will be called a *microcycle*. Every vector that enters a circuit will have a leader, its first element, and some number of followers, all the other elements. The CORDIC circuits perform vectoring on the leader and rotation through the same angle for the followers. A vector composed of K elements, for example, will flow into a CORDIC circuit during K consecutive microcycles. The elements making up such a vector will flow out of the CORDIC circuit at the same rate that they entered, one element per microcycle. Although the elements are modified by passage through the CORDIC they retain their identity and their order, and the leader on input becomes the leader on output. The CORDIC circuit has a delay due to pipelining, which is called its latency.

The Q_θ-type transformations are simply the passage of a vector through a CORDIC circuit. The action of the CORDIC circuit on the leader is a phase change such that the leader becomes real. The action of the CORDIC circuit on the followers is to change their phases by the same amount. A CORDIC circuit used in this manner will be called a θ-CORDIC.

While a Q_θ transformation is a phase change applied to a single complex vector, a Q_ϕ transformation is a rotation of pairs of complex numbers through a real angle. To accomplish a Q_ϕ transformation we use two CORDIC circuits, which are called the *master* ϕ-CORDIC and the *slave* ϕ-CORDIC, respectively. The master ϕ-CORDIC deals with real parts of complex data, while the slave ϕ-CORDIC deals with corresponding imaginary parts. Data involved in a Q_ϕ transformation are presented to the two CORDICs as "vectors" of "complex" elements so that the rotation may be accomplished is if it were a phase change; both CORDICs rotate using the same angle. These

vectors are presented sequentially, one element per microcycle, just as for a θ-CORDIC. Refer to Figure 4-1. The input vectors for ϕ-CORDICs have complex elements made up partly from the output of a θ-CORDIC and partly from one column of the Cholesky matrix. The master ϕ-CORDIC gets the real part of its input vector from the real part of the Cholesky matrix column and gets its imaginary part from the real part of the θ-CORDIC output. The slave ϕ-CORDIC gets the real part of most of its input vector from the imaginary part of the Cholesky matrix column and gets the corresponding imaginary parts from the imaginary part of the θ-CORDIC output. However, the leader of the vector that is input to a slave ϕ-CORDIC is a special case, which is discussed below. The outputs of the two types of ϕ-CORDICs are then reassembled. The real parts of the ϕ-CORDIC outputs become the real and imaginary parts, respectively, of the updated Cholesky matrix column. The imaginary parts of the ϕ-CORDIC outputs become the real and imaginary parts, respectively, of the updated tacked-on vector. A more complete explanation follows.

Note that CORDIC circuits are used for two quite different meanings of "rotation," one being the phase modification of a complex number and the other being the coordinate rotation of a two-element vector. In the latter case, although the two coordinates may be complex numbers, the angle through which they are rotated is real, and the same real rotation angle is used on the real and imaginary components, respectively.

First, consider the ith transformation Q_θ. Its input is the tacked-on vector with components $[x_i, x_{i+1}, \ldots, x_N]$, all complex numbers. These are fed into a θ-CORDIC one pair at a time, beginning with the pair $(Re(x_i), Im(x_i))$. The first pair is marked as a "leader" with a special indicator that causes the CORDIC circuit to determine and set the ρ_i controls as it passes through the successive rotation stages (vectoring). Thus the modified x_i that emerges from the θ-CORDIC is rotated in phase so that it is real—its imaginary component is zeroed. The "followers" $x_{i+1}, \ldots, x_N$ are rotated by the same phase as the leader, but they generally emerge from the θ-CORDIC as complex numbers.

It is helpful to think of the tacked-on vector as being pumped through the θ-CORDIC one element at a time, and to think of the CORDIC as a pipelike organ. As many elements as the CORDIC has stages will be inside the pipe at any time.

In the ith transformation Q_ϕ the data being pumped through the transformation are two columns, $[l_{i,i}, l_{i+1,i}, \ldots, l_{N,i}]$ and $[x_i, x_{i+1}, \ldots, x_N]$. These are composed of complex numbers except that the pair $(l_{i,i}, x_i)$ is real. Things are arranged so that $l_{k,i}$ and x_k are available simultaneously. One CORDIC, the master ϕ-CORDIC,[1] gets its two inputs from the real parts of the two input streams. The slave ϕ-CORDIC takes its two inputs (with one exception) from the imaginary parts of the two input streams. The pair $(l_{i,i}, x_i)$ is marked as a leader so that as it passes through the master ϕ-CORDIC it determines the angle controls ρ_i for that CORDIC, which are to be used to rotate all the follower pairs $(Re(l_{k,i}), Re(x_k))$ which pass through the master ϕ-CORDIC. The *same* angle controls are to be used in the slave ϕ-CORDIC to rotate the pairs $(Im(l_{k,i}), Im(x_k))$. The slave ϕ-CORDIC is not needed for $(Im(l_{i,i}), Im(x_i))$ because this pair is $(0, 0)$. Therefore it is "tricked" into setting its angle controls identical to those of the master ϕ-CORDIC by giving it the leader

[1] The terms master and slave are left over from an earlier usage and are not meaningful.

pair $(Re(l_{i,i}), Re(x_i))$ in place of $(Im(l_{i,i}), Im(x_i))$. The outputs of the ϕ-CORDICs are an updated column of the Cholesky matrix and an updated tacked-on vector with its ith element zeroed.

In this way, the use of three CORDIC circuits accomplishes the pair of transformations needed to zero out one element in the tacked-on vector. This three-CORDIC configuration is called a *supercell.* In Figure 4-1 we illustrate how the θ-CORDIC and the master and slave ϕ-CORDICs are interconnected and how the pipelining of each individual CORDIC circuit extends naturally to the pipelining of the supercell.

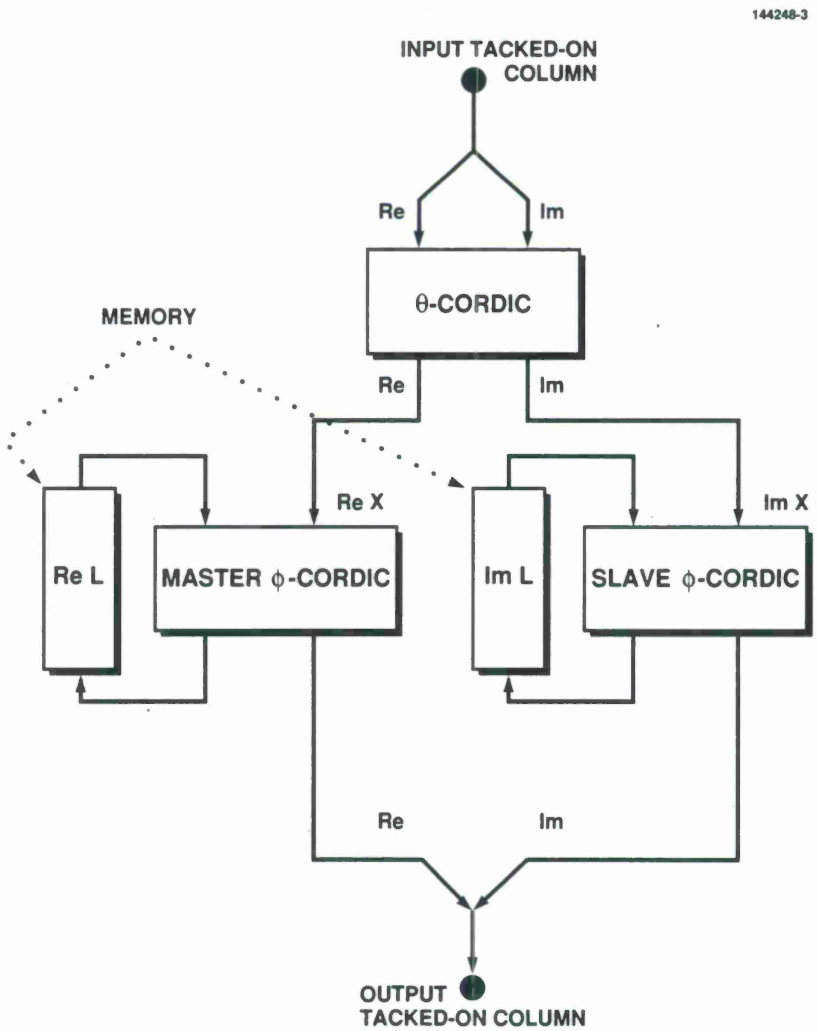


Figure 4-1. A CORDIC *supercell.*

Figure 4-1 shows how the column $[l_{i,i}, l_{i+1,i}, \ldots, l_{N,i}]$ can be stored within the supercell in local memory. On the other hand $[x_{i+1}, \ldots, x_N]$ must be passed to another supercell where it is needed by the $(i+1)$st Q_θ transformation and subsequent Q_ϕ transformation. Since the leader of the

tacked-on vector is zeroed out by passing through a supercell, the element immediately following it in the sequence becomes the new leader element.

4.2 Simple Systolic Array

Now that there is a configuration for a supercell that can update one column of a Cholesky matrix, a complete systolic array using N such supercells can be configured. Figure 4-2 shows such a systolic array, except that the local memory in each supercell, which holds the column that the supercell must update, is shown separately. In this array the data passed from supercell to supercell are components of the tacked-on vector, one complex word at a time. Thus, at the input to the first supercell the data seen, in order of arrival, are:

$$\ldots|, |x_1, x_2, x_3, \ldots, x_{N-1}, x_N|, |x_1, x_2, \ldots$$

Each block consists of one vector sample of observed interference $\underline{X}$. Note that *as soon as one block's vector has been pumped into the first supercell, it is ready to receive another block's vector.* The time interval between the first elements of two successive vectors will be called a macrocycle.

The output of the first supercell is a modified tacked-on vector. Its other product, the updated first column of the Cholesky matrix, is retained within the supercell, to be used as original data for the next update. The modified tacked-on vector has one fewer component than the original. Thus the output of the first supercell (input to the second supercell) takes the form

$$\ldots|, |b, x_2, x_3, \ldots, x_{N-1}, x_N|, |b, x_2, \ldots$$

where the b's are blanks, e.g., time intervals when no data is present. The second supercell's output (the third supercell's input) has two blanks per block:

$$\ldots|, |b, b, x_3, \ldots, x_{N-1}, x_N|, |b, b, x_3, \ldots$$

Strictly speaking, the blanks represent zero-valued samples. But these are not values that just happen to be zero— they are inevitably zero. Thus they need not be explicitly sent from supercell to supercell, nor involved in computation. Therefore the first supercell is busy all the time, the second supercell is idle for one microcycle out of every N, the third supercell is idle for two microcycles out of every N, and the last supercell is idle for all but one microcycle out of every N.

4.3 Latency

Since a CORDIC circuit is pipelined, there will be a circuit latency. This means that there is a delay between presenting an element at the input and getting the corresponding result at the output. Latency is essentially the amount of time the data stays in the "pipe." A supercell has the latency of two CORDIC circuits. We will use τ to represent the latency of a supercell in microcycles.

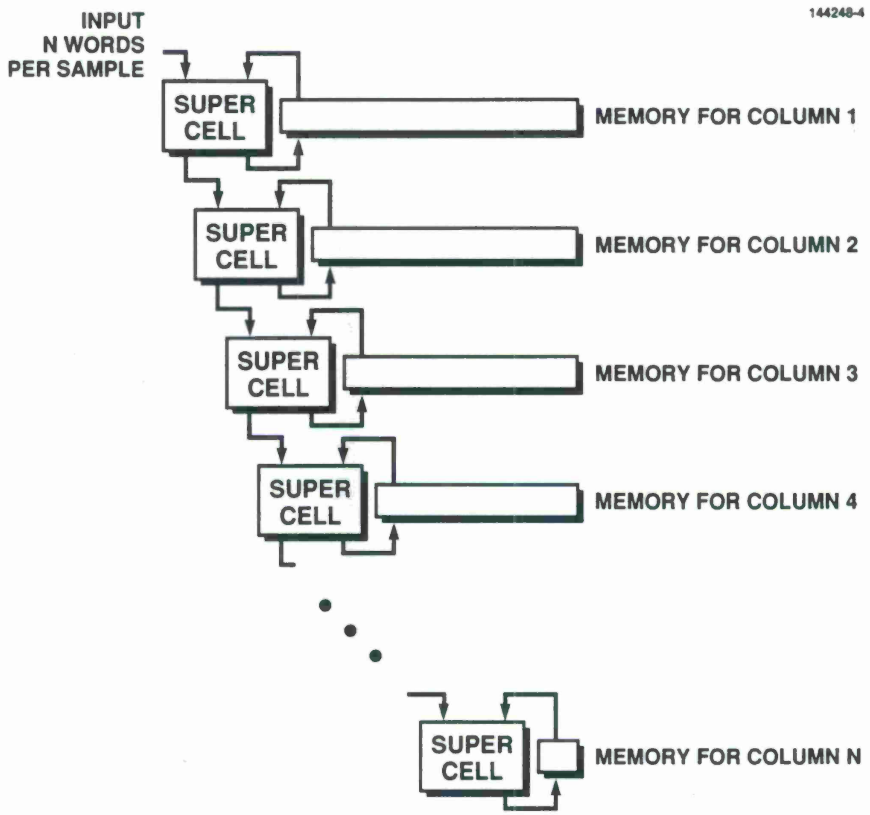


Figure 4-2. Systolic array to update a Cholesky matrix.

Thus as a simple example, if $\tau = 3$ and $N = 5$ then there would be the following time relationship among the five (numbered 0 to 4) supercells:

$$
\begin{array}{rl}
\text{- 0:} & x_1 \;\; x_2 \;\; x_3 \;\; x_4 \;\; x_5 \\
\text{0 - 1:} & \quad\;\; \text{b} \;\; x_2 \;\; x_3 \;\; x_4 \;\; x_5 \\
\text{1 - 2:} & \quad\quad\;\; \text{b} \;\; \text{b} \;\; x_3 \;\; x_4 \;\; x_5 \\
\text{2 - 3:} & \quad\quad\quad\;\; \text{b} \;\; \text{b} \;\; \text{b} \;\; x_4 \;\; x_5 \\
\text{3 - 4:} & \quad\quad\quad\quad\;\; \text{b} \;\; \text{b} \;\; \text{b} \;\; \text{b} \;\; x_5
\end{array}
$$

This is the timing relationship for successive modifications of one tacked-on vector as it is passed from supercell to supercell. However, because each supercell begins to work on another update as soon as it completes the previous update, a much busier timing relationship would actually be observed:

- 0:	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	x_1	x_2
0 - 1:	x_3	x_4	x_5	b	x_2	x_3	x_4	x_5	b	x_2	x_3	x_4	x_5	b	x_2	x_3	x_4
1 - 2:	x_5	b	b	x_3	x_4	x_5	b	b	x_3	x_4	x_5	b	b	x_3	x_4	x_5	b
2 - 3:	b	b	x_4	x_5	b	b	b	x_4	x_5	b	b	b	x_4	x_5	b	b	b
3 - 4:	b	x_5	b	b	b	b	x_5	b	b	b	b	x_5	b	b	b	b	x_5

This illustrates one way that the systolic array is inefficient. The higher numbered supercells have many microcycles in which they are not doing anything. In addition, the local memory used is different in each ϕ-CORDIC since the columns of L are of different length. The cells must either be provided with different amounts of memory, or, if they are all identical, some of them must waste much of their provided memory.

Despite the inefficiency, we should note the excellent characteristics of this architecture for updating a Cholesky matrix:

- The CORDIC circuits are highly specialized for rotation, hence they are efficient.
- The CORDIC circuits are naturally pipelined, hence they may be clocked at high speed.
- Data is passed only locally from processor to nearby processor, hence there is no need for a global bus.
- All CORDIC circuits are operated in synchrony, using the same clocks.
- Data arrives where it is needed just exactly when it is needed, hence there is no need for buffer storage.

4.4 Forgetting Factor

In practice, we want to update the Cholesky matrix as if the data were weighted by an exponential time window. This can be accomplished if each ϕ-CORDIC attenuates its L output slightly by a forgetting factor α before reusing it in the next update. Since the CORDIC must have a correction gain K as well, we propose to realize a single correction gain of $K\alpha$. This choice has some consequences. Because there is some flexibility in the choice of α, we can choose it so that we can multiply by $K\alpha$ with minimum hardware. We were able to use only three adder stages in the eventual choice, $\alpha = 0.99867003$. Each tacked-on vector from n updates earlier affects the Cholesky matrix as if it had been multiplied by α^n. Specifically, α^{320} is approximately 0.6532.

We would prefer that the corrected gain of θ-CORDICs be unity, and that the corrected gain for ϕ-CORDICs also be unity when correcting the gain for the tacked-on vector. However, it simplifies the hardware design if we use only one correction gain. The attenuation of the tacked-on vector due to the forgetting factor, as the tacked-on vector is passed through 128 CORDICs in series, is slight enough that it may be shown to make no difference in nulling performance.

5. AN EFFICIENT SYSTOLIC ARRAY WITH LATENCY CONTROLLED INTERLEAVING

In the previous section a systolic array for updating a Cholesky matrix was introduced. In this section a major improvement that increases the efficiency of the systolic array is explained.

To begin, each supercell was assigned the responsibility for updating not one but *two* columns of the Cholesky matrix. The lengths of the two assigned columns will always add up to $N + 1$.[1] This is done as in Table 5-1.

TABLE 5-1.

Pairing of Columns in the Same Supercell

Supercell	Column No.	Length	Column No.	Length
0	1	N	N	1
1	2	N-1	N-1	2
2	3	N-2	N-2	3
⋮	⋮	⋮	⋮	⋮
k	k+1	N-k	N-k	k+1
⋮	⋮	⋮	⋮	⋮
N/2-1	N/2	N/2+1	N/2+1	N/2

A modification of the simple systolic array to use the pairing leads us to Figure 5-1. By this pairing we obviously equalize the workload of the supercells, since each supercell is required to deal with two leaders and $N - 1$ followers for every new vector sample of interference that comes along. Only half as many supercells are needed as before. Furthermore, the memory, which is used in ϕ-CORDICs to store the columns of the Cholesky matrix to be updated in that supercell, is the same size in all the supercells, $N + 1$ words per ϕ-CORDIC.

This modification allows us to retain all the advantages of a systolic array. There are two aspects of this statement that need clarification. First, because the tacked-on vector in the originally proposed systolic array is passed from supercell $k-1$ to supercell k, and since in the modified systolic array supercell k has the responsibilities of supercell $N - 1 - k$, each supercell must pass data both forward and backward. For example, supercell 2 must pass its output to supercell 3 but, later, when it is serving the role of supercell $N - 3$ it must pass its output to supercell $N - 2$ which is really supercell 1. Yet these communications are still only local. There is still no need for a global bus.

[1] We are assuming that N is even. A similar line of development, not taken in this report, can be worked out if N is odd. We would use the first supercell for the longest column and pair the remaining columns so that their lengths added up to N.

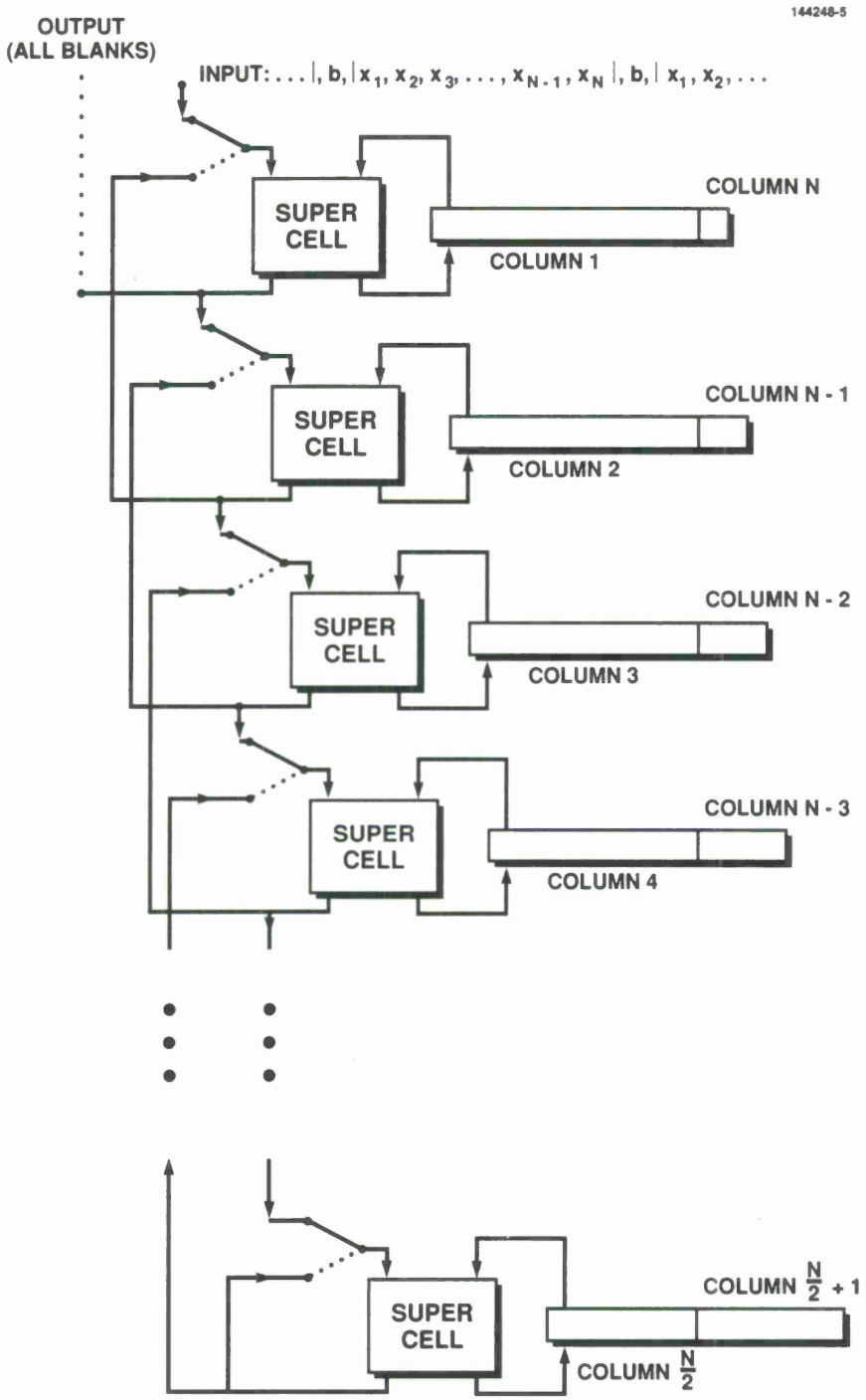


Figure 5-1. Global architecture of folded systolic array.

The second consideration is more difficult. In a systolic array we would like the data to get to where it is needed exactly when it is needed. It would not be suitable if supercell k were passed tacked-on vectors from supercell $k+1$ and also from supercell $k-1$ during the same microcycles. To explore this issue in more detail we need to take account of the supercell latency τ as well as the workload of each supercell. Further, we must consider more than one update cycle at a time. For example, supercell 0 would devote N microcycles to the update of column 1 for the current tacked-on vector, then devote one microcycle to the update of column N for a much older input. In supercell 0, suppose that the current tacked-on vector is input during microcycles $0, 1, \ldots, N-1$. When supercell 0 serves as supercell $N-1$, it works on a tacked-on vector of length 1. Therefore we must leave at least one blank microcycle per cycle. It would be best if the one-element vector arrived exactly as the supercell becomes available during microcycle N. Then the whole timing pattern can repeat periodically beginning in microcycle $N+1$ with a new N-element tacked-on vector during microcycles $N+1, N+2, \ldots, 2N$ followed by a blank in microcycle $2N+1$ during which, as supercell $N-1$, a new one-element tacked-on vector is input.

Letting the latency be an unknown quantity τ, we can deduce when inputs arrive at supercell 1, then at supercell 2, etc., until a pattern is obvious (see Table 5-2). The pattern allows us to write the list for any generic supercell, P_k, and for the generic supercell P_{N-1-k} that is its alter ego. The microcycles when inputs are presented are *congruent* mod $N+1$ to the lists in Table 5-2.

TABLE 5-2.

Microcycles when Each Supercell Input Is Busy

Processor	When Inputs Are Presented	Inputs/cycle
P_0	0,...,N-1	N
P_1	τ+1,...,τ+N-1	N-1
P_2	2(τ+1),...,2τ+N-1	N-2
⋮	⋮	⋮
P_k	k(τ+1),...,kτ+N-1	N-k
P_{N-1-k}	(N-1-k)(τ+1),...,(N-1-k)τ+N-1	k+1

The key point is that τ must be chosen so that all these inputs are presented to supercells during all these microcycles without any data collisions. It is completely adequate to derive a condition on τ from the generic supercell that serves as P_k and P_{N-1-k}. The only legitimate time for supercell P_k to change its role to supercell P_{N-1-k} is just after microcycle $k\tau + N - 1$, when the latter must accept the arrival of data from the other direction, beginning in a microcycle whose number is congruent to $(N-1-k)(\tau+1)$ mod $N+1$.

$$(k\tau + N - 1) + 1 \equiv (N-1-k)(\tau+1) \bmod (N+1)$$

$$k\tau - 1 \equiv (-2-k)(\tau+1) \bmod (N+1)$$

$$k(2\tau + 1) \equiv -(2\tau + 1) \bmod (N + 1)$$

$$(k + 1)(2\tau + 1) \equiv 0 \bmod (N + 1)$$

At this point if the same τ is to work for all k then we must have

$$2\tau + 1 \equiv 0 \bmod (N + 1).$$

The above equation is a congruence. We can write it as an equality with an unknown multiplier m

$$2\tau + 1 = m(N + 1).$$

The smallest allowable[2] latency would be when $m = 1$, namely

$$\tau = \frac{N}{2}.$$

The idea of choosing a latency that allows all the various partially zeroed tacked-on vectors of different lengths to interleave with one another without collisions is called *latency-controlled interleaving.* It allows us to "fold" the linear systolic array, which was only about 50 percent efficient, into a half-size linear array that is 100 percent efficient, without paying a penalty in complication of control.

Earlier, when the inputs to ϕ-CORDICs was discussed, we arranged that $l_{k,i}$, which comes from a local memory, and x_i, which comes from a θ-CORDIC, should arrive at the ϕ-CORDIC input during the same microcycle. This is also accomplished by latency-controlled interleaving, but of a much simpler sort. The delay from one update cycle to the next is $N + 1$ microcycles. Therefore, as $l_{k,i}$ is updated, the updated $l_{k,i}$ must be delayed, by a combination of the latency of the ϕ-CORDIC and some extra delay, so that it appears at the input again $N + 1$ microcycles later. The latency of a ϕ-CORDIC is approximately $\frac{\tau}{2}$. The extra delay needed can be provided by a small memory, which has about $N + 1 - \frac{\tau}{2}$ words.

In Figures 5-1 and 5-2 the folded systolic array is shown. Figure 5-1 is the global architecture and Figure 5-2 shows the communication for a typical supercell and its neighbors. The necessary control of this systolic array has two aspects. First, the data passed from CORDIC cell to CORDIC cell must be marked with an indicator of whether it is a leader or a follower. Second, since a θ-CORDIC cell can take its input from two other supercells, forward or backward in the chain, some control must tell it which to choose.

A simple rule accomplishes both controls. At the input to the first supercell (supercell 0), an input bit called *direction* is provided. When *direction* = *true* the leading supercell takes its input from the given N-element tacked-on vector. When *direction* changes to *false* the leading supercell

[2] Because N is even, a parity argument will show that m must also be odd.

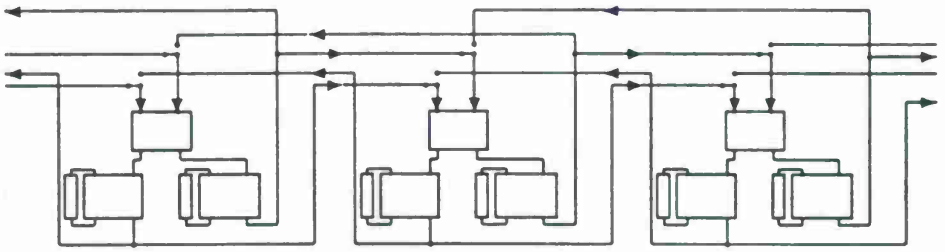


Figure 5-2. Local communication within folded systolic array.

takes its next input from supercell 1 (acting as supercell $N-2$). The change in direction from false to true or from true to false serves as a mark of an element that is a leader, causing the CORDIC to set its controls as that element passes through the stages.

The direction bit is passed systolicly along with the tacked-on vector through both types of CORDIC cells and emerges from the ϕ-CORDICs to be used in the next supercell. However, the leading edge must be delayed by one additional microcycle relative to the tacked-on vector so that the change in direction is attached to the first element *not zeroed out*. The direction bit travels only forward along the chain. Direction bits output during processing of data being passed backward would be redundant and are ignored. In this way, the control of the folded array is actually no more complex than the control of the original systolic array, and each supercell generates the control for the supercells that follow it in the chain.

6. SIMULATION TO SELECT PARAMETERS

We want MUSE to be able to demonstrate adaptive nulling with S/N improvement of at least 50 dB. Now there are many factors in a system that control the achievable improvement in S/N due to adaptive nulling. These include the number and distribution of jammers, their strength in comparison to thermal noise in the system, the antenna element gain patterns, mismatch in the front ends and in all other circuitry prior to digitization, the adequacy of the statistical representation of interference (principally, how many samples used to characterize it) and, finally, the accuracy of the digital computation. MUSE's design should not rely on special knowledge of the environment, the antenna elements, or front ends. The point of view is taken that MUSE should be capable of adapting to artificial random inputs generated with only the constraint that 50 dB of nulling is possible. Even here, however, there is a caveat. Since it is relatively easy to provide artificial data whose dynamic range will overwhelm any fixed design, we must require that the artificial data sets include some bound on the largest jammer to thermal noise ratio, and indeed, this bound should be only a little bit more than the desired 50 dB S/N improvement sought.

With this goal, we have been able to select MUSE parameters by extensive numerical simulation. Each hardware configuration under consideration was simulated exactly at the "bit-level" with the chosen number of stages, word-length, etc. Simulated data sets were contrived with any selected number of "jammers," such that exactly 50 dB of S/N improvement (or any other desired amount) was theoretically possible for that data set. The "condition number" of the contrived data was also controlled so as to be realistic. Except for these two constraints, the data sets were random and several data sets with the same condition number and theoretical S/N improvement potential were tried for each experiment. In appendix A, the mathematical method by which we can produce data from a random scenario with these two constraints is explained. The Cholesky matrix determined by MUSE for each data set was used to determine weights and the weights were used to determine the actual S/N improvement. Therefore, the loss in performance due to the finite word-length, finite number of CORDIC stages, rounding and truncation, etc. was determined.

Based on these simulations, four finite word-length parameters could be determined: the word-length of internal CORDIC registers, the word-length for the tacked-on vector, the word-length for the Cholesky matrix, and the number of CORDIC stages. The "forgetting factor" was also studied and is described elsewhere.

The heart of the simulation is a subroutine that simulates a CORDIC process with integer arithmetic. Because a typical CORDIC stage requires scaling data by 2^{-i} and adding it to unscaled data, the word-length required for a precise execution of the equations in Figure 3-2 would increase the number of bits to the right of the binary point by i bits in the ith stage. Thus we need rounded arithmetic in almost every stage. However, such rounding introduces computational noise in every stage, and the total noise could be excessive. Guard bits introduced at the low end of each word reduce the rounding noise. Experiments showed that two guard bits reduced the rounding noise adequately.

It was also necessary to consider overflow. Of course, overflow in a CORDIC might be inevitable. If x and y are both near their maximum allowed values at input and are rotated by 45°, then a component of the result can exceed the maximum by a factor of almost $\sqrt{2}$. This kind of overflow must be prevented by appropriate scaling of the inputs, a consideration common to all fixed-point signal processing projects. However, a second kind of overflow is also a concern—overflow within an interior CORDIC stage. This was "prevented" in connection with realizing the combination of the CORDIC gain compensation and forgetting factor. The required fixed multiplication (1242/2048) is achieved by using three shift-and-add circuits, e.g.,

$$\frac{1242}{2048} = \frac{(2^7 + 2^3 + 2^1)(2^3 + 1)}{2^{11}}.$$

The multiplication is split between a factor of 138/256 at the input to the CORDIC (the guard bits are created at this point) and 9/8 at the output of the CORDIC (the guard bits are dropped at this point). The reason for splitting the factor in this way is to ensure that overflows can only occur in the CORDIC circuit's interior in cases when overflows would be inevitable due to simply storing the result of an accurate rotation in the available word-length.

Because the simulation of the CORDIC takes exact account of the finite word-length arithmetic process, the results it produces are bit-by-bit identical to the results expected from actual hardware.

The simulated CORDIC subroutine is called with each pair of inputs, and with a flag telling it whether this is a leader pair for which the controls are to be set or a follower pair for which previous controls are to be used. The subroutine is called three times within another subroutine that simulates a supercell for a pair of inputs. An L-Update subroutine calls the supercell subroutine $64+63+62+\ldots+2+1 = 2080$ times for each tacked-on vector. The L-Update subroutine is called once for each vector sample in the data set. This simulation ignores the latency and other timing issues, since they have no bearing on the relations between parameter choices and performance.

The contrived data sets comprised exactly $N = 64$ vector samples for which the ideal Cholesky matrix was predetermined. Ideal weights were computed using floating point arithmetic for this ideal Cholesky matrix to confirm that the predetermined S/N improvement was correct. Then these 64-sample data sets were repeated as MUSE inputs several times in succession, and snapshots of L were taken after every block of 64 samples. The computed Cholesky matrices were then used to compute MUSE weights using accurate arithmetic. Any loss in S/N improvement when the MUSE weights are used in place of the ideal weights could therefore be attributed to the computational error in computing L with finite word-length arithmetic.

Another method of computing weights will be described in section 8 in which the linear equations for the desired weights are solved, after L is already computed, using the same finite word-length CORDIC arithmetic. Therefore nulling based on weights computed that way should generally show an additional loss in S/N improvement. This was also studied.

In Figure 6-1, the abscissa gives the condition number of a possible data set and the ordinate gives the S/N improvement possible for that data set. The solid curve gives the upper bound of the set of possible scenarios. With a given condition number, no data set is possible for which a sidelobe canceller can give an S/N improvement that lies above this curve. Some scenarios marked

as crosses on the plot are illustrated. The most interesting data sets are points just a little below the curve, representing strong jamming that can be deeply nulled.

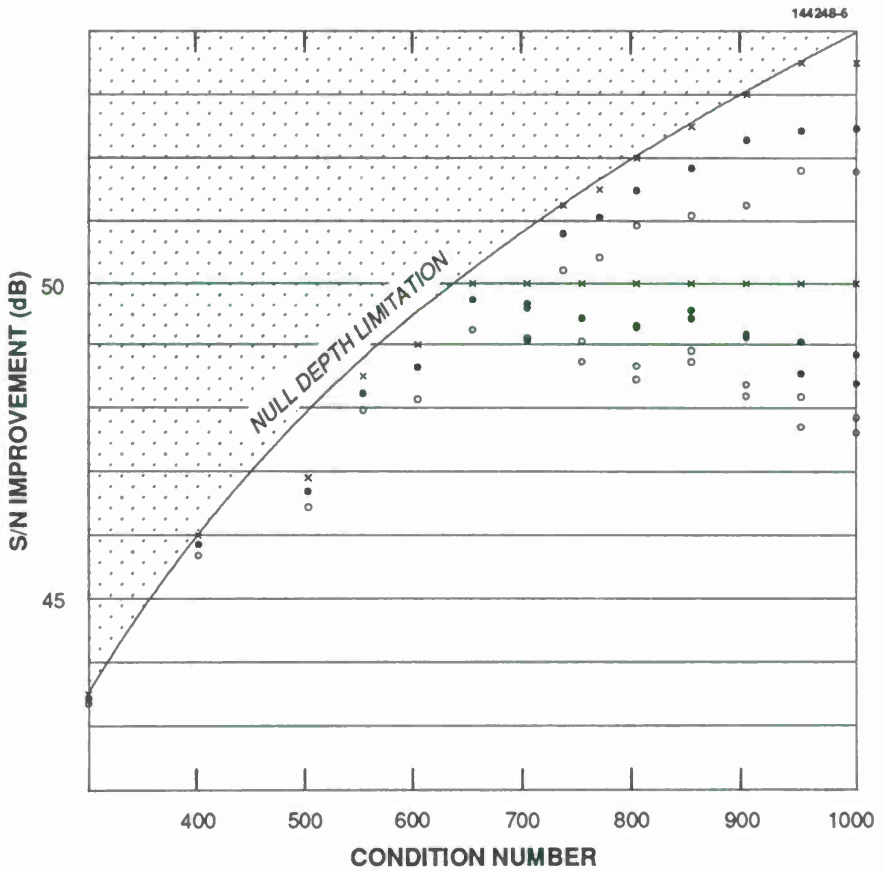


Figure 6-1. Scenarios and performance for ideal nulling and for simulated hardware parameter choices.

Individual experiments with fixed word-length parameters produce reduced S/N improvements. Those weights obtained by accurately solving for weights using an inaccurately computed Cholesky matrix are marked as filled circles on the plot. The S/N improvements for the same inaccurately computed Cholesky matrix when the weights are computed using the technique of section 8 are indicated by open circles. When several scenarios with the same theoretical performance were tried, there is one cross and a scattering of filled and open circles.

The data in Figure 6-1 are for the final parameter choices.

Suppose we are experimenting with the number of CORDIC stages. By making generous choices for the other finite word-length parameters, we can get some idea of how parsimonious we could be

with the number of CORDIC stages. The other finite word-length choices were guessed at similarly. Then, by varying word-lengths, guard bits, and stages, one bit or stage at a time, we could zero in on the best trade-off of hardware cost and performance.

Several observations were clear from studying many cases in this way. First, the loss in S/N improvement due to parsimonious parameter choices increases with the condition number of the data. For a data set with a condition number around 300, the loss is negligible. For a data set with condition number about 1000, the loss is about 2 dB. Therefore, if we want a system to be capable of 50 dB S/N improvement, it would be unwise to try to make parameter choices based on data sets for which the condition number is higher than 850. Second, when one parameter is chosen too parsimoniously, curves that plot performance as a function of another parameter can be too irregular to be useful, often seeming to suggest that saving a bit or a stage is helpful rather than harmful. Third, the loss in S/N improvement does not seem to depend on the number of jammers as long as there are fewer jammers than degrees of freedom. All the experiments illustrated in Figure 6-1 were run using 35 CW jammers.

After many experiments, final parameter choices were made. The number of CORDIC stages, exclusive of the three stages used for the combination of CORDIC correction gain and forgetting factor, was selected to be 13, e.g., $\nu_{max} = 12$. The word-length of internal registers of the CORDIC cell was chosen to be 24 bits. Of these 24 bits the two least significant bits are guard-bits and are not passed beyond the CORDIC. The tacked-on vector data passed between CORDIC cells uses a 22-bit word-length. The stored $l_{i,j}$ are also 22-bit words. Note that the inputs to the wafer, obtained from the antenna elements, although presented as 22-bit words, must not be allowed to use the full 22-bit dynamic range since the Cholesky matrix, generated from many such tacked-on vectors, contains the energy of all of them and so is much larger than any single one.

It was decided earlier that MUSE should be capable of updating its Cholesky matrix about 45,000 times per second. Each vector needs 67 microcycles[1] to be fed into MUSE, so the time for a microcycle must be about 1/3 microsecond. The system clock rate is four times as fast, about 12 MHz.

Statistical accuracy of the weights to within about 1 dB of optimum requires about 320 updates [3]. This means that the weights can be updated as often as 140 times per second. However, MUSE is physically capable of updating its weights much more frequently or less frequently because a "snapshot cycle" is designated by a special external signal.

[1] MUSE accepts a new 64-element vector every 67 microcycles, rather than every 65. This will be explained in section 8.

7. APPLICABILITY OF RESTRUCTURABLE VLSI

The systolic array described in the preceding sections has 32 supercells, each requiring three CORDIC cells. A CORDIC cell can be designed as an integrated circuit. Such a circuit requires some tens of thousands of transistors, classifying it as VLSI. Because the three types of CORDIC cells are so similar to one another, it suffices to design only one type of CORDIC cell, capable of operating in three different modes. The MUSE system can be realized efficiently using an advanced technology called Restructurable Very Large Scale Integration (RVLSI). Lincoln Laboratory has already used this technology to realize six different systems, so the technology is reasonably mature. The RVLSI technology is briefly described.

7.1 Restructurable Very Large Scale Integration

Restructurable VLSI comprises a design methodology, a laser-based interconnect modification technology, and a set of CAD tools for building large area integrated circuits [8,9]. Wafers are fabricated with redundant circuits and interconnect, which are tested after fabrication. A laser is then used both to form and to break connections to the operable circuits, and to build the desired system. The laser can also customize circuitry by, for instance, setting constant coefficients in circuits used to implement a filter function. The size of the basic replaceable unit, the cell, is determined by partitioning considerations and fabrication yield; typically a cell comprises thousands of transistors. Experience has shown that building a wafer with about twice as many cells as ultimately needed strikes a good balance between interconnect overhead and cell yield.

Several laser restructuring technologies have been developed [10]; the technique to be used in the MUSE application, which uses the laser to form a connection between two adjacent diffusions, is completely compatible with standard IC processing. The laser diffused link is used for both signal and power connections. Connections are broken by using the laser to vaporize a metallization path. The connections and the metal cuts are made with high yield and appear to be very reliable.

A set of software tools has been developed for the design tasks that are unique to wafer scale design [11]. *Floorplanner* is used to plan the tiling of a wafer with cells and interconnect and *IRT* performs a unique assignment and routing based on test results for each wafer. *IRT* also adds connections for testing of the wafer during restructuring and allows removal and replacement of circuits that are found to be defective during this process.

Six RVLSI systems have been built on three different wafer designs [8] with the largest one having 405,000 functional transistors. One *Integrator* wafer scale system has been operating in a bench tester for 4 1/2 years without failure. The MUSE system and another now being built [12] are by far the largest and most logically complex wafer scale circuits.

7.2 Adapting MUSE for RVLSI

Several design choices were made in order to make the MUSE design even more compatible with the RVLSI technology.

First, because of the significant cost of intercell connections, the microcycle was divided into four steps so that data could be moved into and out of a CORDIC cell in four small pieces. Consider a θ cell that must read two 22-bit words from either of two sources (the supercell ahead of it and the supercell behind it) while outputting two 22-bit words to the ϕ cells of its supercell. This would have required 132 connections to at least as many metallization lines, without even counting clocks, control, power and ground, and jumpers. The four-step microcycle allows the same data to be moved with 33 connections and 33 metallization lines.

Second, the two adder/subtractors in each CORDIC stage were replaced by one adder/subtractor time-shared between two tasks. It first computes its "y" output

$$y \leftarrow y + 2^i \rho_i x,$$

but preserves the old y for the next computation

$$x \leftarrow x - 2^i \rho_i y.$$

The adder/subtractor is the largest part of the stage, much larger than the control or the pipeline latches. Therefore, this change reduces the size of the stage. In exchange, it takes twice as long to perform a minirotation since this is essentially the time to perform the addition and the subtraction. Since a CORDIC cell, even after this change, is relatively large, and since an adder can be designed adequately fast, this modification was judged worthwhile.

Third, it was decided to provide each CORDIC cell with adequate memory to store the required real or imaginary part of the two columns of the Cholesky factor, even though the θ-CORDIC cells have no need for this memory. The memory uses only a minor portion of the silicon area of each cell, and the cost of this silicon area must be balanced against the competing cost of providing connections between the ϕ-cells and a second type of cell, "memory," or alternatively, against the cost of making two types of CORDIC cells, some with memory and some without.

The fourth change is very much more complex and is dealt with in section 8. Briefly, after enough samples have been used to update a Cholesky matrix and the time has come to use that matrix to solve the linear equations for the nulling weights, it would have been necessary to provide data paths to pass the $N(N+1)/2$ matrix elements off the wafer to where they could be used next. Instead, we found a way to use the CORDIC cells to begin the process of solving the linear equations. This method produces $2N$ intermediate results, which are the only quantities that must be sent off the wafer, and they are moved along on the paths already provided for moving the tacked-on vector.

Finally, we will make the CORDIC cells in two mirror image versions and run the discretionary metallization paths between them. The ability to use CORDIC cells on either side of a "bundle" of

discretionary connection tracks gives extra flexibility. Details of the RVLSI CORDIC cell are given in section 9, and details of the wafer floorplanning and restructuring are given in section 10.

7.3 Relationship of Cell Yield to System Yield

In this subsection it is assumed that the process by which a wafer is manufactured has a significant probability of defects. Assume we make K cells per wafer. Assume that each cell on the wafer functions correctly as a CORDIC with a probability p_c, independent of whether its memory works, while each cell's memory is perfect with probability p_m, independent of whether it is a part of a functional CORDIC cell or not. Therefore a cell chosen at random is usable as a θ-CORDIC with probability p_c and is usable as a ϕ-CORDIC with probability $p_c p_m$.

Assuming that we have adequate testability and connectability, we can compute the probability, P_w, of forming a completely functional MUSE system on a wafer in terms of K, p_c, and p_m. A given wafer is suitable if at least 96 CORDIC cells work and if, among the CORDICs that work, at least 64 have working memory. The probability that exactly i of K cells have CORDIC function is

$$p_{i/K} = \binom{K}{i} p_c^i (1-p_c)^{K-i},$$

and the probability that exactly i working CORDICs have exactly k working memories is

$$p_{i/K:k/i} = \binom{K}{i} p_c^i (1-p_c)^{K-i} \binom{i}{k} p_m^k (1-p_m)^{i-k}.$$

P_w can therefore be written as

$$P_w = \sum_{i=96}^{K} \binom{K}{i} p_c^i (1-p_c)^{K-i} \sum_{k=64}^{i} \binom{i}{k} p_m^k (1-p_m)^{i-k}.$$

This formula can be evaluated by a computer, but it is useful to first make a well-studied approximation [7]. The expected number of working CORDICs per wafer will be $N_c \equiv Kp_c$ and its variance will be $\sigma_c^2 = Kp_c(1-p_c)$. Then

$$p_{i/K} = \binom{K}{i} p_c^i (1-p_c)^{K-i} \approx \frac{\exp -\frac{(i-Kp_c)^2}{2Kp_c(1-p_c)}}{\sqrt{2\pi Kp_c(1-p_c)}}.$$

The approximation is valid when $Kp_c(1-p_c) \gg 1$. Both the approximation and the correct formula give nonnegligible results except when i is within about five standard deviations of its mean.

In a similar fashion the expected number of working memories among i CORDICs is $N_m(i) \equiv ip_m$, with variance $ip_m(1-p_m)$. Then for $p_{i/K:k/i}$ the appropriate approximation is

$$p_{i/K:k/i} \approx \frac{\exp -\frac{(i-Kp_c)^2}{2Kp_c(1-p_c)}}{\sqrt{2\pi Kp_c(1-p_c)}} \frac{e^{-\frac{(k-ip_m)^2}{2ip_m(1-p_m)}}}{\sqrt{2\pi ip_m(1-p_m)}}. \tag{7.1}$$

This approximation is valid when $Kp_c(1-p_c) \gg 1$ and $kp_m(1-p_m) \gg 1$.

Then we may conveniently compute P_w as

$$P_w = \sum_{i=96}^{K} \frac{\exp -\frac{(i-Kp_c)^2}{2Kp_c(1-p_c)}}{\sqrt{2\pi Kp_c(1-p_c)}} \sum_{k=64}^{i} \frac{e^{-\frac{(k-ip_m)^2}{2ip_m(1-p_m)}}}{\sqrt{2\pi ip_m(1-p_m)}}. \tag{7.2}$$

A computer program to evaluate this double sum may be further optimized by limiting the summation indices. For the outer sum the low index is the greater of 96 and $Kp_c - 5\sqrt{Kp_c(1-p_c)}$, and the upper index is the lesser of K and $Kp_c + 5\sqrt{Kp_c(1-p_c)}$. Similarly, for the inner sum, the low index is the greater of 64 and $ip_m - 5\sqrt{ip_m(1-p_m)}$ while the upper index is the lesser of i and $ip_m + 5\sqrt{ip_m(1-p_m)}$.

In Figure 7-1 the solid family of curves give wafer yield, for a wafer design using 130 cells, versus CORDIC cell yield, with different assumed values of memory yield. The curves, from right to left, are for memory yields of 50, 60, 70, 80 and 90 percent (the curves for 80 and 90 percent are too nearly identical to be resolved).

Of course, only a limited number of wafers will be made, probably ten. It is not difficult to compute P_{bw}, the probability that the best of the ten wafers will have enough useful cells. The probability that a given wafer is inadequate is $1-P_w$ and the probability that all ten are inadequate is $(1-P_w)^{10}$. Hence the probability that at least one out of the ten wafers can be used for a wafer scale MUSE is

$$P_{bw} = 1 - (1-P_w)^{10}.$$

The dotted curves in Figure 7-1 give the probability of at least one usable wafer versus CORDIC cell yield for the same choices of memory yield (curves for 70, 80, and 90 percent are almost indistinguishable).

An examination of the dotted curves in Figure 7-1 shows that memory failures of up to 30 percent make almost no difference in the expected wafer yield or probability of success. Based on a CORDIC yield of about 70 percent, about what we have with test cells, we would expect about an 85 percent chance of project success in a run of ten wafers. However, this probability drops sharply with decreasing CORDIC yield.

If we have insufficient CORDIC cell yield to make a complete MUSE system on a single wafer, we can still make a wafer scale system by linking together two wafers. Only a few connections need to be made between the two wafers because a relatively small number of connections run from supercell to supercell. Roughly speaking, two wafers would have twice as many CORDIC cells to start with and therefore should allow system success with about half the yield that would be required with a single wafer. In reality the situation is slightly more favorable since the lower

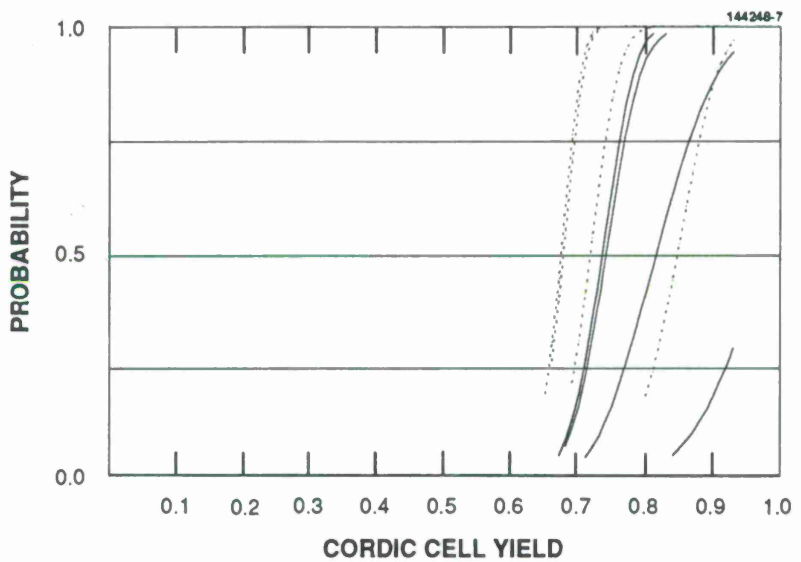


Figure 7-1. Expected wafer yield versus CORDIC *cell and memory yield.*

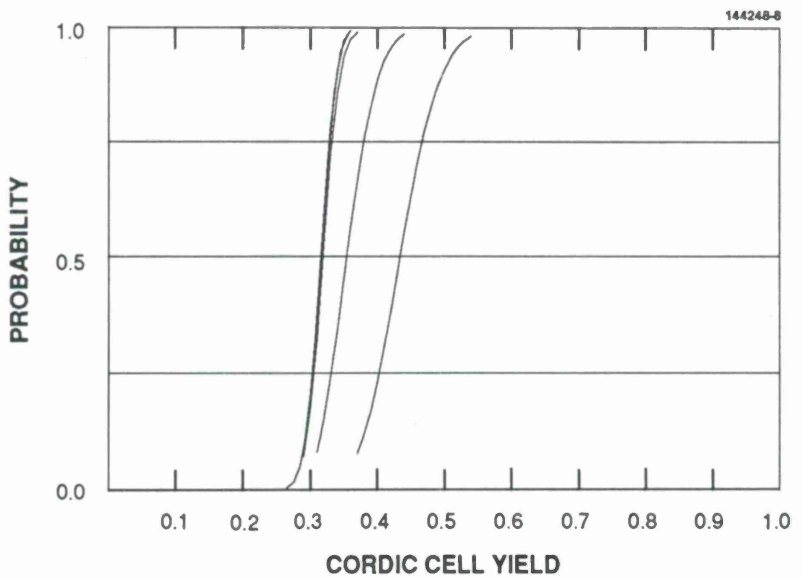


Figure 7-2. Expected system yield versus CORDIC *cell and memory yield, using best two of ten wafers.*

CORDIC yield implies a greater variation from wafer to wafer and since we would use the best two wafers. The probability of forming a MUSE system from the best two wafers out of ten is given by:

$$
\begin{aligned}
P_{2w} = & \sum_{n_1=0}^{K} \sum_{n_2=0}^{K} \sum_{n_3=0}^{K} \cdots \sum_{n_{10}=0}^{K} S(\mu(n_1, n_2, n_3, \ldots, n_9, n_{10})) \\
& \binom{K}{n_1} p_c^{n_1} (1-p_c)^{K-n_1} \binom{K}{n_2} p_c^{n_2} (1-p_c)^{K-n_2} \cdots \\
& \binom{K}{n_{10}} p_c^{n_{10}} (1-p_c)^{K-n_{10}} \sum_{m=64}^{\mu(n_1, n-2, \ldots, n_{10})} \binom{\mu}{m} p_m^m (1-p_m)^{\mu-m}
\end{aligned} \tag{7.3}
$$

where

$$
\mu(n_1, n_2, n_3, \ldots, n_9, n_{10})) = \max_{i \neq j}(n_i + n_j),
$$

and

$$
S(m) = \begin{cases} 1 & m \geq 96 \\ 0 & otherwise \end{cases}
$$

In Figure 7-2 we plot the probability of finding 96 working CORDIC cells with 64 working memories on the best two out of ten wafers. The curves, from right to left, are for memory yields of 50, 60, 70, 80, and 90 percent (curves for 80 and 90 percent are indistinguishable). From Figure 7-2, we would predict that with memory yield of only 70 percent and CORDIC yield of only 34 percent we have an 85 percent chance of forming a MUSE system with two wafers out of ten.

8. USING CORDIC CELLS TO SOLVE LINEAR EQUATIONS FOR WEIGHTS

8.1 Mathematical Preliminaries

Consider the general $N \times N$ set of linear equations

$$A\underline{X} = \underline{B},$$

where A and $\underline{B}$ are given and $\underline{X}$ is to be determined. To solve the linear equations, we will introduce a seemingly unrelated second problem. First, construct an $N \times (N+1)$ matrix by appending $\underline{B}$ onto the right edge of A.

$$[A \mid \underline{B}].$$

Next, postmultiply it by an $(N+1) \times (N+1)$ unitary matrix Q that causes the last column of this matrix to become zeroes.

$$[A \mid \underline{B}]\, Q = \left[\hat{A} \mid \underline{0}\right].$$

Third, partition Q as

$$\left[\begin{array}{c|c} & \\ Q_{uu} & \underline{Q}_r \\ & \\ \hline \underline{Q}_l^h & q \end{array}\right].$$

From this partition we can obtain the equation

$$A\underline{Q}_r + q\underline{B} = \underline{0}$$

leading to

$$A\left(\frac{-\underline{Q}_r}{q}\right) = \underline{B}.$$

This shows that the solution to the original problem, solving for $\underline{X}$, is hidden in the elements making up the last column of the unitary matrix Q in the second problem. If we can zero out a column $\underline{B}$ tacked on to a matrix A using a unitary transformation Q,[1] then we can solve $A\underline{X} = \underline{B}$. But the collection of CORDIC cells that update a Cholesky matrix are performing just such a unitary

[1] Actually, there is no requirement that Q be unitary.

transformation. This suggests that we might use the same CORDIC cells to solve for weights $\underline{W}$ using that Cholesky matrix as the matrix of coefficients in the linear equations.

Note that in section 2 it was noted that there are generally two sets of linear equations to solve, the first set involving L and the second set involving L^h. But in the case of a sidelobe canceller (a particular choice of the steering vector, $\underline{S} = [0, 0, \ldots, 0, 1]^t$), the first set can be solved by inspection rather than by computation, so that only the second set, that involving L^h, needs to be solved using the CORDIC cells.

The first set of equations to be solved was:

$$\begin{aligned} l_{11}y_1 &= s_1 = 0 \\ l_{21}y_1 + l_{22}y_2 &= s_2 = 0 \\ l_{31}y_1 + l_{32}y_2 + l_{33}y_3 &= s_3 = 0 \\ \vdots &\quad \vdots \\ l_{N1}y_1 + \ldots + l_{NN}y_N &= s_N = 1 \end{aligned}$$

We can successively see that $y_1, y_2, \ldots, y_{N-1}$ are 0. The last equation in the set reduces to

$$l_{N,N}y_N = s_N = 1,$$

so that $y_N = 1/l_{NN}$. In short, the vector $\underline{Y}$ that solves $L\underline{Y} = \underline{S}$ for the sidelobe canceller case is $\underline{Y} = \frac{1}{l_{N,N}}\underline{S}$. No actual computing steps are needed—the answer is available by inspection.

However, we are still required to carry out computations to solve for $\underline{W}$ in the second set of equations $L^h\underline{W} = \underline{Y} = \frac{1}{l_{N,N}}\underline{S}$. Note that we can replace the scale factor $\frac{1}{l_{N,N}}$ by any other which suits us, since changing it would only scale $\underline{W}$ by a constant, which cannot affect the resulting signal-to-interference ratio.

In the following discussion, we will assume that in each supercell the two columns of the snapshot of L for which we want to solve for $\underline{W}$ are stored in a snapshot memory. The process of solving for the weights can therefore be interleaved with updating the Cholesky matrix as more antenna data is fed into MUSE.

To use CORDIC cells to solve $L^h\underline{W} = \underline{S}$, we begin by transforming the equation set into one involving a lower-triangular matrix with a tacked-on vector. Define the "reversal" matrix J:

$$J \equiv \begin{bmatrix} & & & & 1 \\ & & & 1 & \\ & & \cdot & & \\ & \cdot & & & \\ \cdot & & & & \\ & 1 & & & \\ 1 & & & & \end{bmatrix}.$$

If we premultiply a vector or matrix by J, then the result is to reverse it top to bottom. If we postmultiply a matrix by J, then it is reversed left to right. Note that JJ is an identity matrix. Now we manipulate $L^h\underline{W} = \underline{S}$, first by inserting JJ between L^h and $\underline{W}$; next by premultiplying by J to give

$$(JL^hJ)(J\underline{W}) = J\underline{S}.$$

And finally by conjugating the entire equation

$$(JL^tJ)(J\underline{W})^* = J\underline{S}.$$

This equation is of the form $A\underline{X} = \underline{B}$ where

$$A = JL^tJ,$$

$$\underline{X} = (J\underline{W})^*,$$

$$\underline{B} = J\underline{S},$$

and the form of A is a lower triangular matrix:

$$A = \begin{bmatrix} l_{N,N} & & & & \\ l_{N,N-1} & l_{N-1,N-1} & & & \\ \vdots & \vdots & \ddots & & \\ l_{N,2} & l_{N-1,2} & \dots & l_{2,2} & \\ l_{N,1} & l_{N-1,1} & \dots & l_{2,1} & l_{1,1} \end{bmatrix}.$$

8.2 Description of the Q-operation

Of course, if we solve for $\underline{X}$ we immediately get $\underline{W}$ since $\underline{W} = J\underline{X}^*$. The tacked-on vector $\underline{B}$ is $[1, 0, 0, \dots, 0]^t$. Changing $L^h\underline{W} = \underline{S}$ into $A\underline{X} = \underline{B}$, an equation involving a lower triangular matrix, has been purely an exercise in notation. No actual computations are involved. We shall now show how to solve $A\underline{X} = \underline{B}$ in the array of supercells *without moving the Cholesky matrix out of the supercells in which it is already stored.* This process is called the *Q-operation.*

It is easy enough to begin the process. The first step is to postmultiply the two columns made up of $\underline{B}$ and the last row of L by a unitary matrix which zeroes out the leading element of $\underline{B}$.

$$\begin{bmatrix} l_{N,N} & 1 \\ l_{N,N-1} & 0 \\ \vdots & \vdots \\ l_{N,2} & 0 \\ l_{N,1} & 0 \end{bmatrix}$$

$l_{N,N}$ is available in supercell 0, where it was created in its guise as supercell N-1. That supercell should also have available the fixed constant $b_1 = 1$. The supercell determines its Q-operation rotation controls (Q-op controls) in both its θ-CORDIC and its ϕ-CORDICs using these two quantities as column leaders. These Q-op controls determine the operations performed on the remainder of the two columns. But now we see what looks like a problem. The next step involves the pair $(l_{N,N-1}, 0)$, but $l_{N,N-1}$ is not available in supercell 0. It is, however, available in supercell 1, which already has a connection to supercell 0. We can pass the Q-op controls ρ_i from supercell 0 to supercell 1 using only local connections and then carry out the postmultiplication which originated with the pair $(l_{N,N}, 1)$ in supercell 0 on the corresponding pair $(l_{N,N-1}, 0)$ in supercell 1. Then we pass the same Q-op controls to supercell 2 and apply them to the pair $(l_{N,N-2}, 0)$. We can move the Q-op controls down the chain of supercells into supercell 31 in this way and then back up the chain all the way back to supercell 0. After they have been used in supercell 0 on the pair $(l_{N,1}, 0)$, the Q-op controls are sent *off the wafer*, where they are stored for later use.

As a result of all this, the first column of A and the tacked-on vector $\underline{B}$ are modified. No further use of the first column of A will be made. Indeed, we don't need to save it as it is computed. The tacked-on column, $\underline{B}$, has been modified so that its first value is zero. The remaining elements of $\underline{B}$ (which were zero) are now nonzero values $[b_2, b_3, \ldots, b_N]^t$ which are available in supercells $1, 2, \ldots$, respectively, where they were created by the rotations just discussed.

The next step is to postmultiply the second column of A and the modified tacked-on vector by a unitary matrix to zero out the next element of the modified tacked-on vector.

$$\begin{bmatrix} l_{N-1,N-1} & b_2 \\ l_{N-1,N-2} & b_3 \\ \vdots & \vdots \\ l_{N-1,2} & b_{N-1} \\ l_{N-1,1} & b_N \end{bmatrix}$$

Both elements making up the first pair, $(l_{N-1,N-1}, b_2)$, are available in supercell 1, b_2 having just been computed there. These are the leaders used to set up Q-operation rotation controls in supercell 1. These Q-op controls are moved to supercell 2 where they control the CORDICs acting on the pair $(l_{N-1,N-2}, b_3)$, then on to supercell 3 where they control the CORDICs acting on the pair $(l_{N-1,N-3}, b_4)$, etc. Ultimately these Q-op controls also turn around at supercell 31 and travel up the chain and are sent off the wafer at supercell 0. These Q-op controls are also stored for later use. At the end of this step the elements of the now twice modified tacked-on vector $[b_3, b_4, \ldots]^t$ are available in supercells $2, 3, \ldots$, exactly where they will be needed to carry out the third step. (The modified second column of A is no longer needed.)

A step during which Q-op controls are determined by vectoring is called a *master Q-operation.* The steps during which these angles are used in CORDIC cells downstream are called *slave Q-operations.*

Using this procedure, we can carry out the entire process of zeroing out the tacked-on vector using CORDIC rotations of data *that is never moved from supercell to supercell.* The only information

that moves during the Q-operation is the Q-op rotation controls. These are passed only locally, to the adjacent supercells, so no global communication paths are needed. Each set of Q-op controls is ultimately passed off the wafer, where they are to be used in the final phase of weight determination.

In this final phase, the elements in the last column of the unitary matrix Q must be determined. Although we have used the CORDICs to postmultiply $[A \mid \underline{B}]$ by Q, Q was never determined in the standard matrix form. The Q-op controls which have been passed off the wafer are all that is needed to instruct CORDIC cells to carry out the operations which realize a postmultiplication by Q. Therefore one can use any CORDIC cells with these same controls to postmultiply *an identity matrix* by Q. This will give the elements of Q, which can be used directly as the desired weights in the nulling problem.

8.3 Summary of Revised MUSE Control and Timing

The systolic array presented in earlier sections was 100 percent efficient. Therefore there can be no time available for the CORDICs to carry out the Q-operation. It was decided to interleave the steps of the Q-operation with subsequent updating of the Cholesky factor, by providing two "idle" microcycles in every macrocycle. It is also necessary to revise the latency to compensate for the longer macrocycle. Here the revised timing is summarized.

The fastest clock present in the system is called a *half-step*. A half-step is the time needed to pass 11 bits between CORDIC cells. Two consecutive half-step intervals make a *step* and suffice to move a word. The reciprocal of the step interval is the rate at which new data is presented to an adder within any CORDIC stage. Two steps are called a *microcycle*. A microcycle is the time required to input one complex element into any θ-CORDIC, and its reciprocal is the rate at which new rotations may begin in any CORDIC cell.

The time interval between input of successive 64-element vectors into MUSE is 67 microcycles, called a *macrocycle*. At the input to any θ-CORDIC cell the 67 microcycles are assigned to move data as described in section 5 except that a blank microcycle is inserted between each tacked-on vector. These blank microcycles are used to move Q-op controls.

Any macrocycle can be designated as a "snapshot" cycle. When the Cholesky factor has been updated to reflect the effect of the vector fed into MUSE during a snapshot cycle, the Q-operation is initiated. The interval between successive snapshot cycles is called an update cycle.

The timing of MUSE all builds up from the half-step. The CORDIC cell designed is capable of being clocked at 12 MHz. Table 8-1 summarizes the various clock speeds and periods implied by the 12 MHz half-step clock.

Since MUSE has 96 CORDIC cells, each performing three million new rotations per second, a conventional computer requiring ten instructions per rotation would have to perform 2.88 billion instructions per second to perform MUSE's task.

The revised latency of a supercell, in units of microcycles, must be

$$\tau = \frac{N}{2} + 1.$$

TABLE 8-1.

Projected Speed of MUSE

Interval		Duration	Rate
Half-Step		83 1/3 ns	12 MHz
Step	= 2 half-steps	166 2/3 ns	6 MHz
Microcycle	= 2 steps	333 1/3 ns	3 MHz
Macrocycle	= 67 microcycles	22 1/3 μs	44.78 kHz
Update Cycle	assuming 300 macrocycles	6.7 ms	150 per second

9. RVLSI CELL DESCRIPTION

The CORDIC cell has been fabricated in $2\mu m$ CMOS and tested at the design speed. It amounts to 54,000 transistors and is 5.5 $mm^2 \times 5.6$ mm^2 excluding output drivers and bond pads. This section briefly describes the cell layout. It consists primarily of a large datapath, two static RAMs, and a chunk of control logic.

9.1 Cell Datapath

The main datapaths and functional blocks of the CORDIC cell are shown in Figure 9-1. Eleven-bit data comes from the forward and reverse connections at the top. It is turned into 22-bit words and passed into the input switching section, then down through the CORDIC stages and output switching before being split into 11-bit words again. The delay memory is provided for the ϕ-CORDICs to delay L_i; the snapshot memory stores one complete L for use in the Q-operation. Along the right edge, the intracell serial communication path is shown; this carries information during the Q-operation.

Inside the input switching box are several delay registers, two shift registers, two adders, and a big multiplexer. The shift registers receive serial data from the serial path and can insert it into the main datapath; this function is used by the θ-CORDIC cells for the Q-operation. The gain compensation is split into three adders; the first two are in the input section to scale down the input number slightly (by the factor 138/256). This ensures that preventable overflows are avoided.The multiplexer allows one of six of data values to be entered into the datastream going to the CORDICs: the input, the memory output, either serial register, the constant 0, or the constant 2^{18}. The last two are only used by the Q-operation.[1]

The basic structure of the CORDIC stage was described in section 3; the sharing of one adder between real and imaginary parts of a word is described in section 7.2. The resulting logic consists of three registers, a one-of-two multiplexer, and an add/subtract unit. The CORDIC stages are 24 bits wide; 13 stages are required to meet the accuracy criterion described in section 6.

Within the output switching box are several delay registers, one serial register, and one adder. The serial register puts data onto the internal serial path during the Q-operation in the ϕ- CORDICs. The adder completes the gain compensation (with a gain of 9/8). The delay memory simulates a long shift register with a circular buffer. Address generation is provided by a simple counter that resets to 0 upon reaching 51; each location is first read, then written. The snapshot memory stores two L vectors acquired at different times as outlined in section 5. Its address generation is more complicated and is described in the next subsection.

[1] Since the input and output word-lengths of a CORDIC cell are the same, overflow cannot be made impossible. For example, if the real and imaginary inputs are approximately equal and the angle of rotation is near to $-45°$, we expect the real part of the output to exceed either input by about $\sqrt{2}$. While overflow for all possible inputs cannot be prevented, we can insist that overflows occur only when the correct output data would not be representable in a 22-bit word.

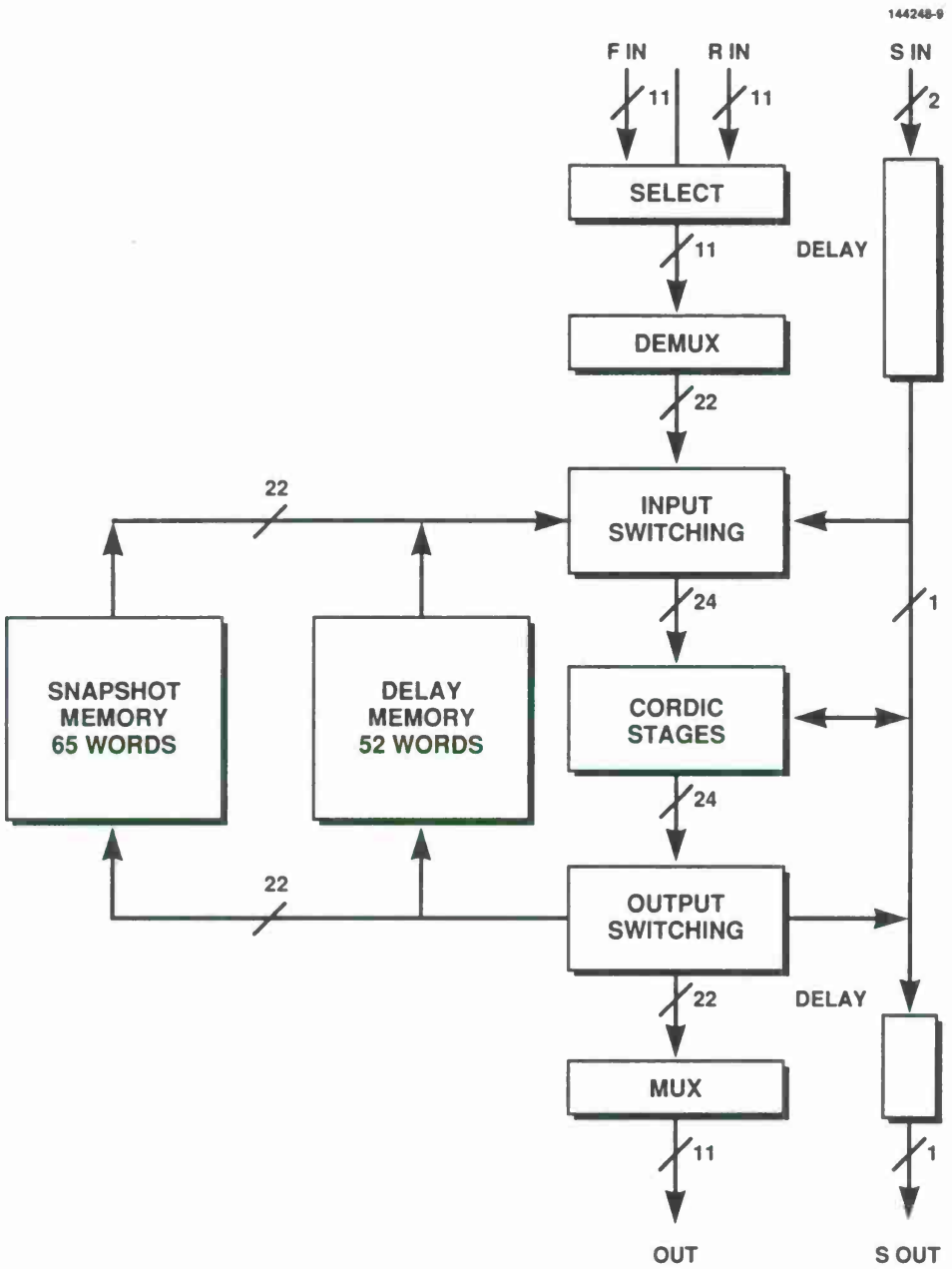


Figure 9-1. Datapaths of CORDIC *computation chip.*

The serial path is used for a number of purposes, depending on whether the cell is a θ cell or a ϕ cell. In all cases it is only used during the Q-operation. Q-op control data is passed from supercell to supercell during the "blank," but within the supercell the blank contains rotation data. Therefore, the θ cell receives parallel format angle values and passes them serially to the ϕ cells. The ϕ cells must output the angle value in parallel format. During a slave Q-operation, each CORDIC stage must read one of the angle bits, while during a master Q-operation each stage must set one of them. The internal serial path provides all of this capability; the control section ensures that the operations occur at the right time.

9.2 Control

MUSE is a systolic array and therefore depends on local communication for control. As mentioned in earlier sections, there are two control signals: D for direction control, and Q to indicate when a snapshot should occur. Q may reach a θ cell from either the forward or the reverse direction. All control is generated from these two inputs, plus one global input, *ZERO*. One update cycle consists of the following: a snapshot in the forward direction, a snapshot in the reverse direction, $N-2$ slave Q-operations, and two master Q-operations. The Q-operations occur in both forward and reverse directions and may be interleaved.

The outside world designates the beginning of a forward snapshot by asserting the Q line to the first cell in the chain. The particular column of L next updated, called a snapshot vector, will be operated on by each cell, and then *copied into the snapshot memory.* Each cell sees this snapshot vector in two pieces, first in the forward direction and then in the reverse direction. Each part of the vector is a different length as described in section 5, but the sum of the two lengths for a given cell is always N+1. The forward vector is copied into the snapshot memory from address 0 upwards, while the reverse vector is copied into the memory from address N downward.

Each supercell in the chain follows the same rules for the Q-operation: when a forward Q-operation is received, use the data pointed to by the reverse pointer and decrement the pointer. If the reverse pointer is 0, generate a forward master Q-operation in the next blank. When a reverse Q-operation is received, use the data pointed to by the forward pointer and increment the pointer. If the forward pointer is N, generate a reverse master-Q operation in the next reverse blank. Whenever a master Q-operation is generated, pass a slave Q-operation on down the chain. Always pass along a slave Q-operation.

Once the reverse snapshot has finished, Q-operations may occur. In the first supercell, the reverse pointer is 0 because its forward snapshot is N words long. Therefore, it generates a forward master-Q operation as soon as the reverse snapshot is done, and passes a slave Q-operation to the second supercell. The second supercell, whose reverse pointer was 1, decrements it to 0 and then generates a master Q-operation. This pattern continues down the chain and back up again, so that the first supercell finally carries out 63 slave Q-operations followed by its second master Q-operation. One Q-op control value is passed off the wafer in the reverse blanks for each reverse Q-operation the first supercell carries out.

10. WAFER SCALE CONNECTIONS

MUSE wafer wiring is simply an iteration of the connections for one supercell shown in Figure 10-1, plus certain peripheral circuits and connections. Note that there are three general classes of connection nets. The first comprises data and the associated D and Q control signals, a bundle of 11 to 13 conductors; the second, the serial paths between rotator cells of a supercell; and the third, global clock and control signals, plus power. The bundled and serial classes together can be described as "interconnections," in contrast to global connections. The RVLSI CORDIC cells have interconnection ports along one edge, global ports along the opposite edge. They are placed as mirrored images between alternating types of wiring channels, corresponding to the interconnections and the global connections.

10.1 Interconnection

The bundled interconnections, which are a realization of the local communication of Figure 5-2, are illustrated in a simplified form in Figure 10-2. The signal paths are serpentine, with a left-to-right flow in one channel, right-to-left in the next, with each bundle represented as a single track. The figure shows the interconnections of eight supercells, indicated by the subscripts, assembled from 24 CORDIC cells. As on a real wafer, assignment is constrained by cell yield. Some cells, such as those marked with central crosses in Figure 10-2, cannot be used, and others, with defective memories corner crosses can be used only for the θ function.

Note that in the upper channel the "F," "R," and "OUT" ports are in the order of forward signal flow; if one has complete freedom of cell assignment, as for supercells 0 and 1, only three tracks are necessary. In the lower channel the port order is contrary to signal flow, and four tracks are required, as for supercells 4 and 5. In order to use cells with defective memory as θ cells, it is often necessary to use an extra track, as shown for supercells 2 and 3 of the forward-flow channel and 6 and 7 of the reverse-flow channel, so that as many as five tracks may be required. In experiments with a number of randomly placed defective cells, the *IRT* routing tool [11] never required more than five tracks, so this can be taken as a practical upper limit.

Actual interconnect is, of course, for bundles, not single tracks. The interconnect channel must accommodate five 13-track bundles, as many as five more tracks for serial signals, a test track for proof of restructuring connections, and spares for possible defective tracks. Therefore, a channel of 85 tracks was chosen, with the connections to the CORDIC cells by an array of five link fields of the type shown in Figure 10-3, with potential links shown as circles. These links can connect the tracks to cell ports, above and below the channel, via short vertical tracks known as stubs. The stub labeling is consistent with Figure 10-1 except for the stubs labeled *XIN, XIN2,* and *XOO,* which are spare ports for *SIN, SIN2,* and *SOO.*

The top track (labeled "16") is for connection testing, and the remaining 16 are for interconnect. The data, Q, and D signals have links in a pair of chevrons, spaced three tracks apart. Primary links for these 13 signals are in the lower chevron, spares in the upper. The final three spares are

tracks 13 to 15. Tracks 14 and 15 are primary tracks for intrasupercell serial signals, and track 13 is a spare.

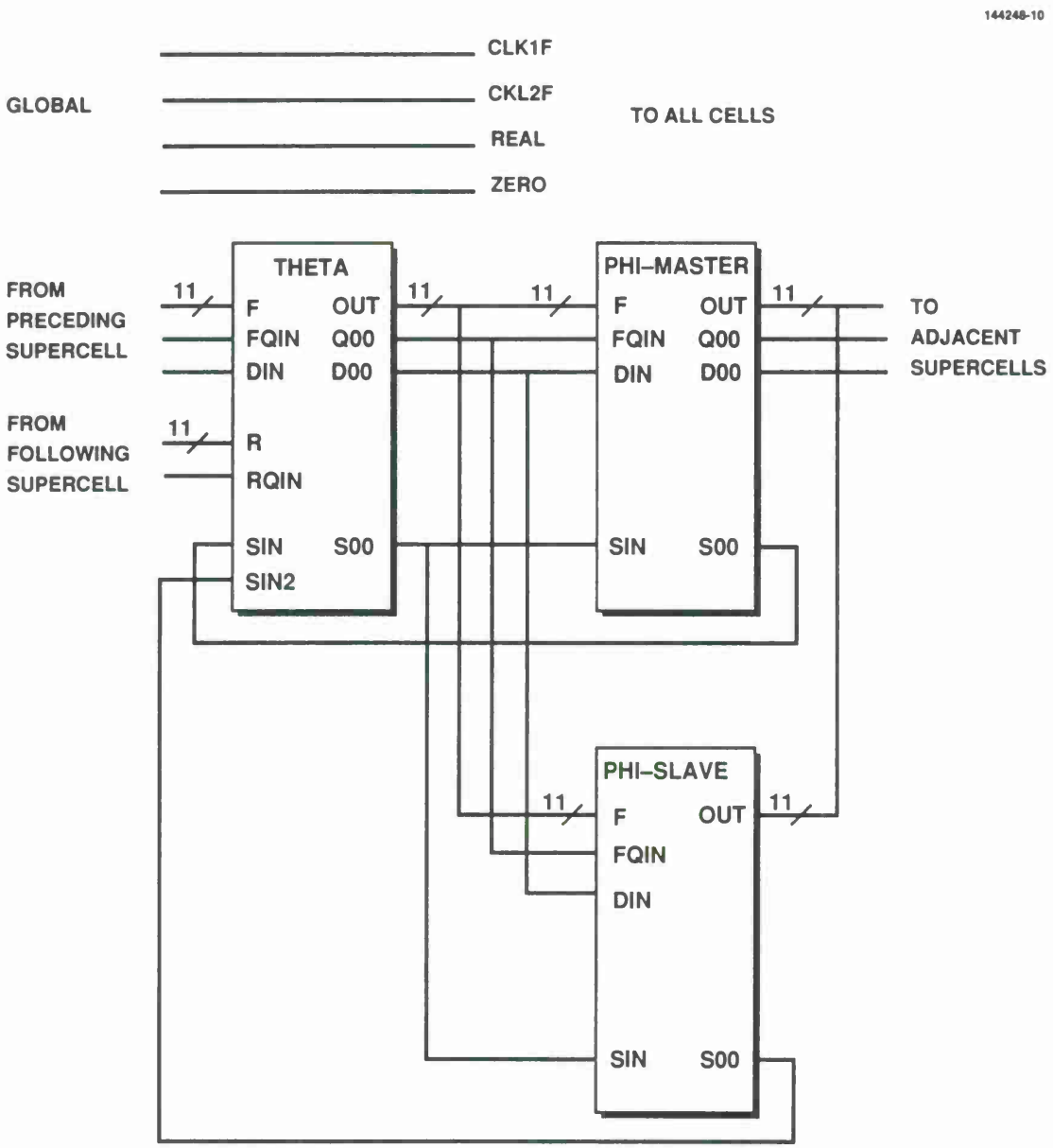


Figure 10-1. MUSE supercell connections.

10.1.1 Channel Defect Capacity

Each signal track can quite naturally be assigned to one of three classes, according to the modulo 3 value of its position in a 17-track subarray, as in Table 10-1.

A worst-case connectivity requirement uses five bundles of tracks per channel. Without counting the tracks used for serial signals within a supercell, this will use up 25 tracks in class 0, out of thirty available, and 20 tracks in each of classes 1 and 2, with 25 of each available. Therefore a channel may have up to five defective tracks per class. However, after consideration of the serial signal track requirements, a channel must not have more than a total of ten defective tracks.

10.1.2 Assignment and Routing

Assignment is the selection of particular RVLSI cells to serve as specified processors; routing is the selection of specific nets of track segments to interconnect the assigned cells. Assignment for MUSE is straightforward, with cell order pretty much following the sequence of the logical description of the wafer. However, to minimize track requirements it is important in the "contra-flow" channels to have the θ cell of a supercell to the left of its ϕ cells.

A MUSE wafer can be routed with a minimum number of tracks if a specific algorithm is used. The basic idea is to proceed in the sequence the tracks appear in Figure 10-3, bottom to top: All data bit "0" nets are routed first, then data-bit "1," and so forth through the Q and D nets. For each signal, primary tracks are used first; secondary ones are used only if a primary is not available. Also for each signal the longest nets are joined first, i.e., those with four ports, then three, and finally two. Serial-signal nets are joined last, again with longer ones preceding shorter. If no track linking to an "S" stub is available the corresponding "X" stub may be used instead, as they are connected within the cell.

10.1.3 Global Connections

Global connections are much simpler than interconnections because every active cell is connected in almost exactly the same way. Restructuring links are used for three purposes. First, the power connections are supplied with power-decoupling capacitors, which are separately tested. Those not shorted are connected to the *Vdd* bus by "zapping" large (200 μm, 4 ohm) links. Second, there are separate power supply connections for the cell memories and the data path. Third, there are spare tracks available for the global signals *ZERO, REAL, CLK1F*, and *CLK2F*. [1]

10.2 Testing During Restructuring

Two types of testing are concurrent with wafer restructuring. The first is a continuity test for link formation. Every CORDIC cell port has an associated photodiode. When an interconnection

[1] These power links are placed within the cells.

TABLE 10-1.

Class Assignment of Tracks for Defect Avoidance

Class	Bundled Signals	Tracks per Channel
0	Data 0,3,6,9; S	30
1	Data 1,4,7,10	25
2	Data 2,5,8; S	25

net is formed, one of the stubs of the net is also linked to a test track, number 16 in the link field of Figure 10-3, which is in turn connected to a wafer-edge pin. The photodiodes of the net can then be consecutively illuminated by a low-power laser beam while photocurrent is sensed at the pin. Upon completion of the test the stub is cut away from the test track.

The second test is of functionality of each cell as it is added to the circuit. For the MUSE wafer, this means adding to the circuit the three CORDIC cells of a supercell, and then applying stimuli to this added supercell while observing signal outputs. The technique is illustrated in Figure 10-4, which shows a step-by-step formation of the eight supercell simplified "wafers" of Figure 10-2.

Step A Three test bundles (shown as tracks) are linked up to provide access to all supercell inputs.

Step B The last supercell of the chain (supercell 7 in this example) is connected, with its output and reverse input brought out to wafer-edge pins, and its forward input connected to one of the test bundles. This single supercell is tested.

Step C The next-to-last supercell is connected and tested in conjunction with the first, with forward input connected to a different test bundle. The test bundle used in Step B is now re-used as a permanent connection from the outputs of the two ϕ-CORDIC cells of supercell 6.

This procedure is repeated for each supercell in turn, until all have been connected. The important point is that the test bundles use only track segments that will eventually be used in the interconnect itself. At each stage in Figure 10-4, new interconnect is formed by cutting off segments of the old test bundle. The only extra wafer wiring resources required are an additional set of wafer-edge pins at the beginning of the chain.

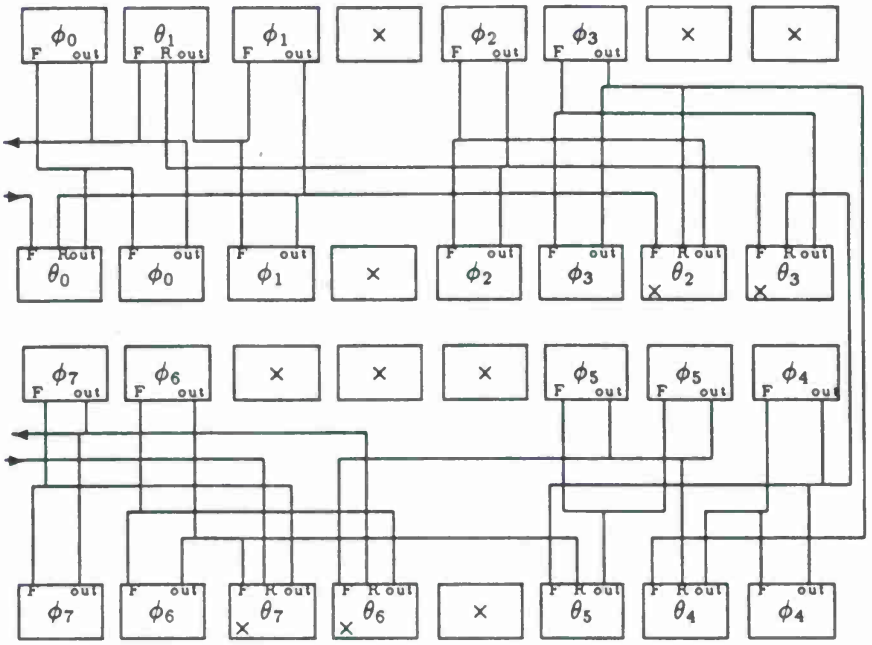


Figure 10-2. Symbolic bundled interconnections of eight MUSE supercells in two wiring channels.

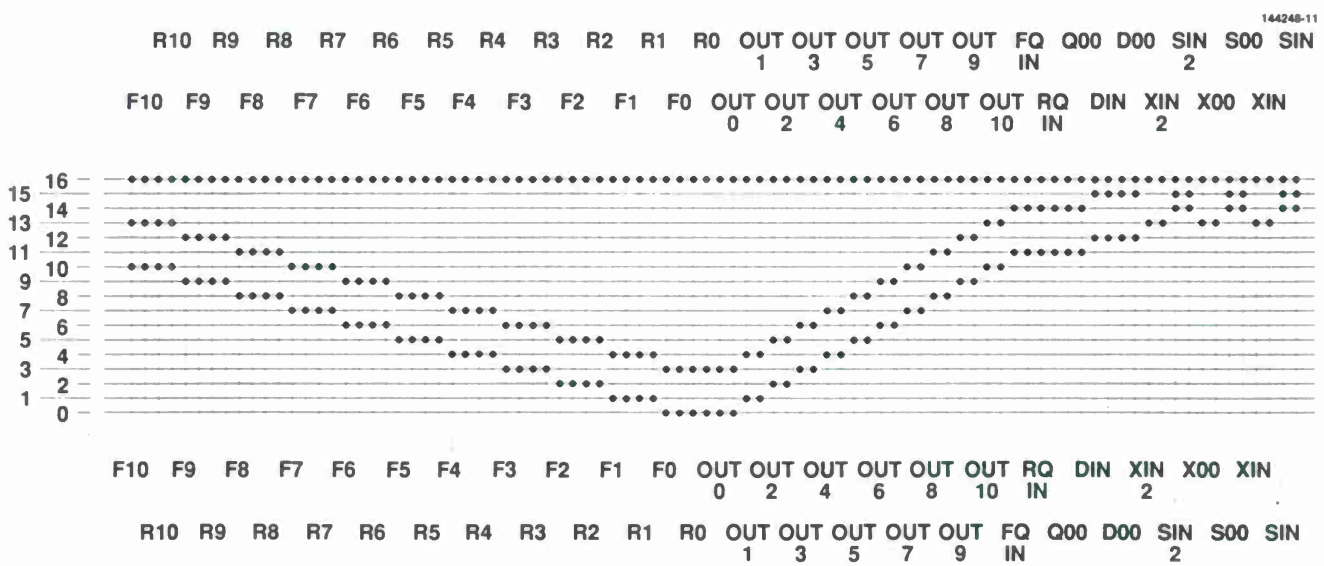


Figure 10-3. Link field for an interconnect channel. The stubs are labeled for their eventual cell ports above or below the channel.

Figure 10-4. Step-by-step connection of supercells of a MUSE wafer. The active test track at each step is marked by a heavy line; inactive ones by light lines.

11. MUSE INVESTIGATION TEST EQUIPMENT (MITE)

11.1 Overview

A test and demonstration system (MUSE Investigation Test Equipment) has been developed to permit us to investigate and demonstrate the performance of the MUSE wafer. MITE can send inputs to MUSE at full speed and collect the angle data representative of the resulting weights. Extensive software permits MITE to generate test data sets, interpret and display the results as antenna patterns or nulling S/N improvement, and compare the results with results of a bit-by-bit simulation of the wafer.

The primary objective of MITE is to test the full wafer. But prior to actual wafer fabrication, MITE has been used to test individual CORDIC cells in a subscale processor that will be discussed in appendix B. This subscale test is important for several reasons:

- It verifies the correct operation of MITE software and hardware.
- It verifies the correct operation of multiple CORDIC cells operating together in the MUSE configuration and with the clocks and controls of the final wafer scale system.
- It verifies the concept of operating a subset of supercells in the manner in which they will be tested during the restructuring of the MUSE wafer, as described in section 10.3.

The subscale tests for $N = 8$, using 12 CORDIC cells, have verified all aspects of the system designs of both MUSE and MITE.

MITE allows an investigator to create sets of N-element data vectors drawn from a common statistical ensemble, representing signals that could have been observed given a specific antenna geometry and jammer scenario. Once the data sets have been created, they are stored in a memory, from which they may be sent into MUSE at a high speed in any order specified by the investigator. The investigator may create several different data sets, representing different jammer arrangements, to examine the performance of MUSE as the statistics of the input data changes with time. MITE is also able to initiate snapshot cycles in MUSE and obtain the resulting Q-operation angle controls, which are captured in a second memory. The MITE then converts the Q-operation angle controls into a corresponding weight vector and can generate graphical displays of the resulting performance of MUSE.

Figure 11-1 shows the MITE test system with a four-supercell subscale MUSE configuration under test.

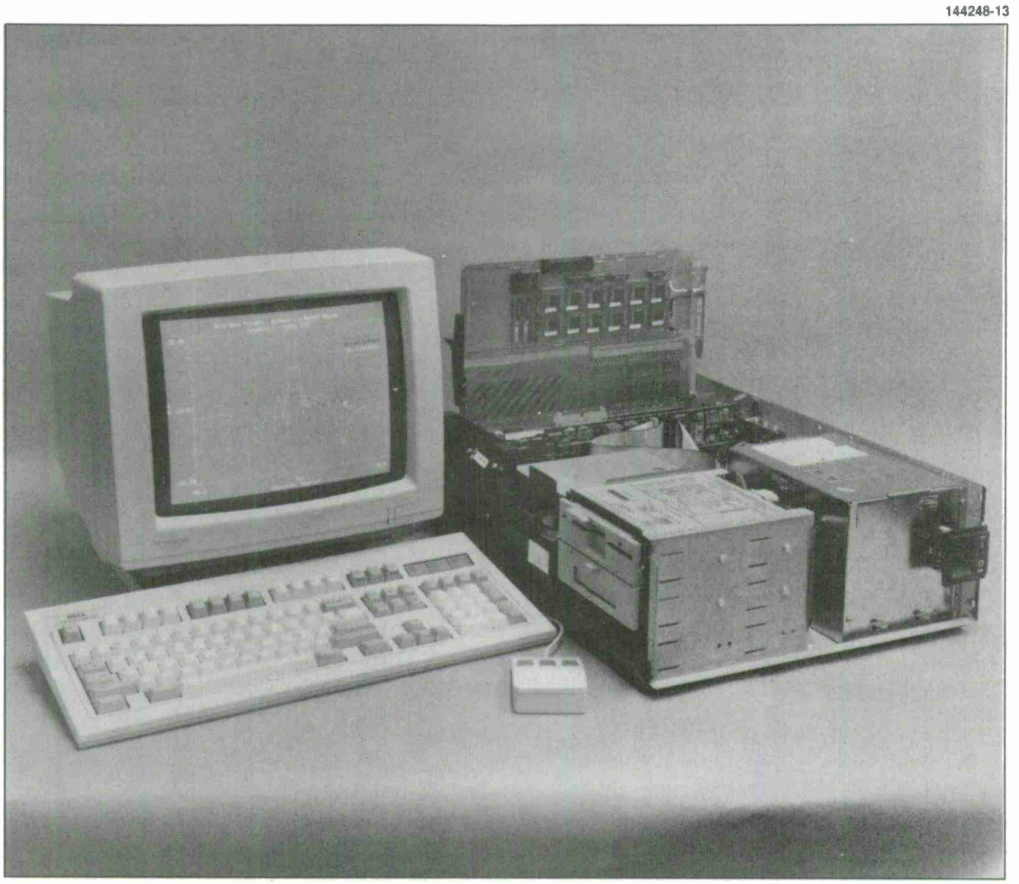


Figure 11-1. Photograph of MUSE Investigation Test Equipment (MITE).

11.2 MITE Architecture

Figure 11-2 shows a block diagram of the MITE. A PC/AT computer is used as the host of the system. The PC/AT is responsible for generating input data, converting Q-operation angle controls to weight vectors and generating displays and performance measures. The interface system consists of two large memories and their respective controls. The interface system has the responsibilities of supplying MUSE with complex data samples, initiating snapshot cycles, and gathering the resulting weight vector information from MUSE.

11.3 Input Data Memory and Control

The data input system has three control signals to MUSE: D, Q, and *ZERO*. Their respective functions are described in detail in section 9. The PC/AT uses an interrupt once per macrocycle from the interface system to control Q and *ZERO*. The direction control, D, is generated by the control logic.

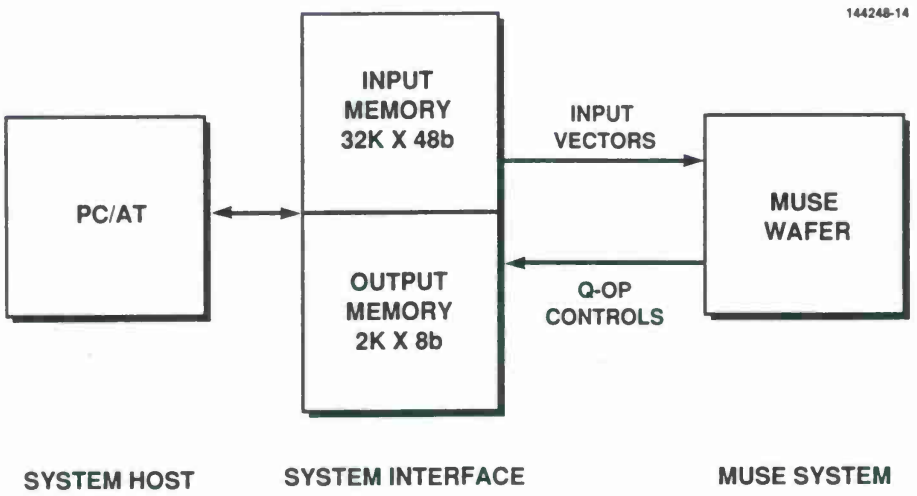


Figure 11-2. MUSE Investigation Test Equipment.

The input memory consists of a 32k by 48-bit static memory. This memory is divided into 512 vectors of 64 complex elements. Each memory location corresponds to one complex element of a vector. The upper nine bits of the address specifies an entire vector, and the lower six bits correspond to a complex element within that vector. Both the PC/AT host and the MUSE interface system have access to the memory. The PC/AT is able to take control of the memory's data and address busses and write data vectors into the appropriate sections of the memory.

The addressing of the memory by the interface hardware is controlled by a DSP56001A microprocessor and a counter. The counter generates the lower six bits of the address, which must be incremented every microcycle. The DSP56001A is used to generate the upper nine bits of the address, which may be changed in a more intricate manner, but only once per macrocycle. During initialization, information is sent from the PC/AT to the DSP56001A that specifies the location of each data set in the memory, the order in which the data sets are to be used and the number of vectors to use from each data set.

11.4 Output Data Memory and Control

The output memory and buffers are shown in Figure 11-3. The latches latch the four 11-bit output values from MUSE during a microcycle in which output data is present, indicated by the Q strobe from MUSE. Then at a much slower rate, the data is written into the FIFO. A counter is used to count the write strobes and interrupt the PC/AT when a complete set of angle words has been accumulated and is ready for the PC/AT to read.

The output data control signals are generated in set of PALs.

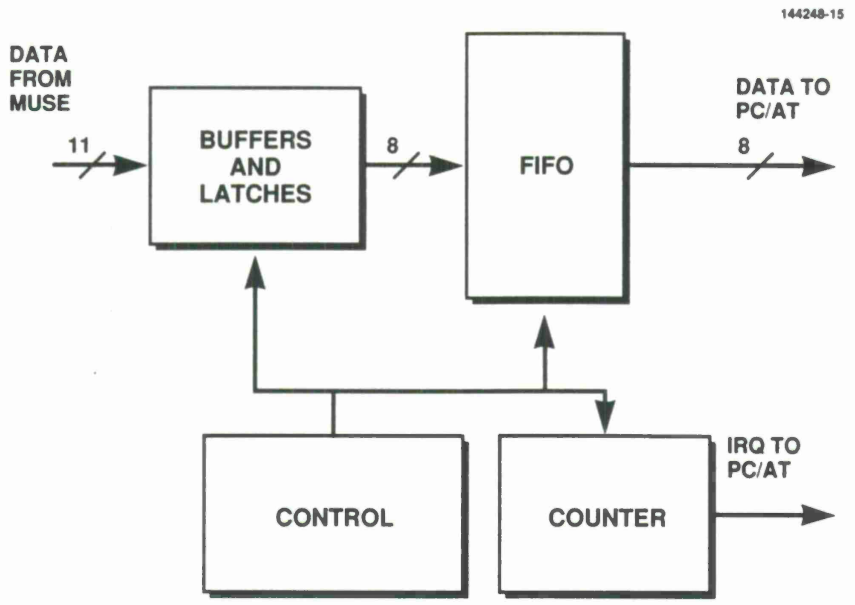


Figure 11-3. Output memory and control.

11.5 User Interface Software

11.5.1 Pull-down Menus

The software system is completely menu-driven. The system allows the investigator to create, edit, view, and load the different types of parameter files used by the system. These parameter files are stored on the hard drive of the PC/AT that allows the parameter files to be used at a latter date.

11.5.2 Scenario Generation

The investigator may create jamming scenarios in two ways. The first type of data set is based on antenna patterns. The investigator can create a parameter file describing a simple main-beam antenna, specifying its beam width and sidelobe levels, and another parameter file with the locations and gains of the omnidirectional auxiliary antennas,[1] and yet a third parameter file that specifies the directions and powers of multiple jammers. Using these three parameter files, the PC/AT can create a scenario file, which is a collection of vectors drawn from a population with statistics appropriate to the jamming and antenna geometry described.

Alternatively, the investigator may create a scenario file with no necessary physical significance, but well suited to testing the fundamental S/N improvement limitations of MUSE without regard

[1] These may be specified individually or placed randomly by the PC/AT.

to the specifics of the antenna geometry or jammer locations. The mathematical description of this type of scenario is described in appendix A.

11.5.3 System Operation

Scenario files must be loaded into the input memory. The investigator may select any desired scenario files from the list of all previously created scenarios files, using the menu system. Next the investigator loads an addressing information file, containing information that tells the system in what order to use the scenario files, and how many vectors to use from each set.

The investigator must then load a file containing a list of snapshot cycles. These are the macrocycles in which the system will order MUSE to start the process to solve for the weight vector. The system will produce a weight vector for each snapshot cycle in the file.

The investigator is then able to start the system and then view the results. The setup information and resulting weight vectors are written to a results file.

The performance of the weight vectors may be displayed in two different formats. The first format, only valid for scenarios created with the antenna pattern, shows the original unnulled antenna pattern, the nulled antenna pattern, and an optimally nulled antenna pattern. This plot is repeated for each weight vector in the results file. The optimal pattern is the pattern which would have been obtained if the weight vector had been solved using the ideal Cholesky factor and using floating point Givens rotations to solve for the weight vector. The second plot shows the S/N improvement vs time. In this plot, the S/N improvement is calculated for each weight vector using the appropriate scenario matrix, and then the S/N is plotted vs the cycle number.

11.5.4 Simulation

The software includes a bit-by-bit simulation of the hardware to allow the results of the MUSE system to be checked for accuracy. Once the user has specified the addressing information, the snapshot cycles and the scenarios, a simulation of the MUSE may be performed instead of running the data through MUSE. The simulation produces the same type of results file as discussed above, and therefore, the results may be viewed using the methods discussed above.

12. SUMMARY

This report has described the complete design of a wafer scale adaptive nulling processor called MUSE, whose individual cells carry out coordinate rotations using the CORDIC algorithm. The CORDIC cells are almost 100 percent utilized and communicate with one another in a simple systolic fashion. The high efficiency and modularity of the algorithm is achieved by interleaving two data streams travelling in opposite directions, and depending on a careful choice of CORDIC latency to avoid collisions.

A CORDIC cell including memory requires approximately 54,000 CMOS transistors and occupies a 5.5 $mm \times 5.6$ mm rectangle. It has been clocked at 12 MHz, so that an entire system of 96 such cells can update a 64-element weight vector for 300 observations in 6.7 ms. A conventional computer would need to be capable of 2.88 Giga-ops to carry out the same task. Simulations have demonstrated that the system can support 50 dB of S/N improvement. A yield analysis has demonstrated that the entire MUSE system can be realized on a single wafer if cell yield averages 70 percent. Furthermore, the simple systolic communication permits using two or more wafers if cell yield is too low.

The CORDIC cells' main function is to update the Cholesky factor of the correlation matrix of observed interference as new data is observed, with a new update approximately every 22 μs. The solution of a set of linear equations whose coefficients are a snapshot of said Cholesky factor gives the desired nulling weights. The CORDIC cells are also used, in an interleaved fashion, to solve these linear equations.

A short chain of twelve CORDIC cells, packaged as chips and representing one-eighth of the system to be realized on a wafer, has demonstrated the correct operation of the cells, their systolic communication, system timing, and S/N improvement, in short, all aspects of MUSE performance. The wafer floorplan and a strategy for testing the wafer as it is restructured are completely designed.

APPENDIX A
CREATION OF DATA SETS WITH CONTROLLED CONDITION NUMBER AND ACHIEVABLE NULLING PERFORMANCE

Let the data set we want to create be designated as U, a matrix with M columns and N rows. Its columns are the vectors we will play into MUSE. Any nonsingular matrix may be expressed in a singular value decomposition

$$U = E\Sigma V$$

where E is an $N \times N$ unitary matrix, V is an $M \times M$ unitary matrix and Σ is an $N \times M$ rectangular diagonal matrix with positive elements, $\Sigma_{ii} \equiv \sigma_i$. Also,

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_N > 0.$$

In what follows, the matrix V will have only a very small role.

The correlation matrix for this data set is

$$R = E\Sigma^2 E^h = E \begin{bmatrix} \lambda_1 & & & & & \\ & \lambda_2 & & & & \\ & & \lambda_3 & & & \\ & & & \cdots & & \\ & & & & \lambda_{N-1} & \\ & & & & & \lambda_N \end{bmatrix} E^h,$$

with $\lambda_i = \sigma_i^2$.

One of the quantities we want to control is the condition number of the data set. The condition number is $\frac{\sigma_1}{\sigma_N}$. We can always scale U appropriately so that $\sigma_N = 1$. Without loss of generality, assume that we have done so. Therefore, in addition, $\lambda_N = 1$.

Once a weight vector $\underline{W}$ is found, by whatever means, ν can be calculated, which is the improvement in S/N for the steering vector $\underline{S}$. Here is how we might do so for the special case of a sidelobe canceller, $\underline{S} = [0, 0, \ldots, 1]^t$. The before signal power is $\underline{S}^h\underline{S} = 1$. The before noise power is $\underline{S}^h R\underline{S}$. The after signal power is $|\underline{S}^h\underline{W}|^2 = |w_N|^2$. The after noise power is $\underline{W}^h R\underline{W}$. Thus the S/N improvement for a general weight vector is

$$\nu = \frac{\underline{S}^h R\underline{S}}{\underline{W}^h R\underline{W}} |w_N|^2.$$

The optimum weight vector is the solution to

$$R\underline{W} = k\underline{S}$$

where k is any constant. Hence

$$\nu_{opt} = \frac{\underline{S}^h R \underline{S}}{w_N^* k} |w_N|^2, and$$

$$\nu_{opt} = \underline{S}^h R \underline{S} \frac{w_N}{k}.$$

In this last equation, w_N depends on k. We can choose a value of k which makes w_N come out to be 1. In terms of the singular value decomposition of U, this value is

$$k = \frac{1}{\sum_{i=1}^{N} |e_{Ni}|^2 / \sigma_i^2}.$$

Similarly

$$\underline{S}^h R \underline{S} = \sum_{i=1}^{N} |e_{Ni}|^2 \sigma_i^2.$$

Therefore

$$\nu_{opt} = (\sum_{i=1}^{N} |e_{Ni}|^2 \sigma_i^2)(\sum_{i=1}^{N} |e_{Ni}|^2 / \sigma_i^2).$$

Now it is possible to choose U such that its singular values are "controlled" and yet consistent with a given amount of S/N improvement.

Suppose only two distinct singular values are used:

$$\sigma_i = \sigma \quad for \quad 1 \leq i \leq K and$$

$$\sigma_i = 1 \quad for \quad K < i \leq N.$$

In other words, we are modeling K equal amplitude and orthogonal jamming sources. The we must have

$$\nu_{opt} = (\sum_{i=1}^{K} |e_{Ni}|^2 \sigma^2 + \sum_{i=K+1}^{N} |e_{Ni}|^2)(\sum_{i=1}^{K} |e_{Ni}|^2 \sigma^{-2} + \sum_{i=K+1}^{N} |e_{Ni}|^2).$$

Abbreviate $\sum_{i=1}^{K} |e_{Ni}|^2$ as q and hence $\sum_{i=K+1}^{N} |e_{Ni}|^2$ as $(1-q)$. Then

$$\nu_{opt} = (q\sigma^2 + 1 - q)(q\sigma^{-2} + 1 - q).$$

Therefore

$$q^2 - q + \frac{1 - \nu_{opt}}{2 - (\sigma^2 + \sigma^{-2})} = 0.$$

The approach will therefore be to choose the desired S/N improvement ν_{opt} and the desired condition number σ. Then we can solve for q, which tells us something about how the energy in the last row of E is distributed. E can then be computed at random subject only to this constraint and to the constraint that it is unitary.

However, not every possible choice of ν_{opt} and σ is compatible. q must be real and positive. The equation for q will give complex roots unless

$$\frac{1 - \nu_{opt}}{2 - (\sigma^2 + \sigma^{-2})} \leq \frac{1}{4}.$$

This equation gives the set of possible scenarios plotted in Figure 6-1. The boundary $\nu_{opt} = \frac{1}{2} + \frac{1}{4}(\sigma^2 + 1/\sigma^2)$ is also plotted in Figure 6-1.

To construct E, begin by constructing its last row. Without loss of generality this last row can be purely real. Begin by constructing N random nonnegative real numbers p_i, $i = 1, \ldots, N$. Normalize the first K so that their sum is q and normalize the last $1 - K$ so that their sum is $(1 - q)$.

$$T_1 = \sum_{i=1}^{K} p_i$$

$$T_2 = \sum_{i=K+1}^{N} p_i$$

$$p_i \leftarrow \frac{q}{T_1} p_i \qquad 1 \leq i \leq K$$

$$p_i \leftarrow \frac{1 - q}{T_2} p_i \qquad K + 1 < i \leq N.$$

Now set

$$e_{Ni} \leftarrow \sqrt{p_i}.$$

Once we have the last row of E, the other rows can be determined sequentially by a Gram-Schmidt orthogonalization procedure. For each new row we generate a completely random complex Gaussian precursor row $\underline{Y}^t$ and subtract off from it any components in the space spanned by the rows already computed. We then normalize what is remaining in $\underline{Y}^t$ to have unit length (unless its length is zero, in which case we must try again with a different random $\underline{Y}$) and this becomes the next row of E. The procedure is continued until we have all N rows of E.

When we have E and Σ (the latter being a diagonal matrix whose diagonal elements are σ and 1), we can multiply them together to give us $U' = E\Sigma$, a matrix with N nontrivial columns that are the MUSE inputs, and $M - N$ columns which are all zeroes. We may interpret the first K columns of U' as "jammers" turning on one at a time, each with energy $\sigma^2 - 1$, and the columns $K + 1$ to N as representing noise only.

It is only at this point that V, the other unitary matrix, is needed. V allows us to distribute the noise and jammer energy defined by U' into M samples in any way we like; by choosing for V we can choose any $M \times M$ unitary matrix, without affecting the two constrained properties of $U = U'V$, its condition number, and the available S/N improvement.

One interesting possibility is to choose V such that U is lower triangular. It then becomes a data set that is its own Cholesky matrix. If $E\Sigma$ is played into MUSE, we expect it to compute L which is approximately equal to that Cholesky matrix.

APPENDIX B
TESTING A MUSE SUBSET

The choice of $N = 64$ substantially impacted the design of the CORDIC cell for MUSE. One impact is due to the relationship between N and latency for latency controlled interleaving. A second impact is in the size of memories built into the chip. For these reasons, the design of the cell dictates the system timing.

For various reasons, it may be desirable to use the CORDIC cell designed for $N = 64$ to realize an adaptive nulling system for smaller N. In the body of this report two such requirements are mentioned: first, the desire to verify the operation of the MITE using CORDIC chips that were fabricated to prove out the cell design; second, the testing of the wafer during restructuring, as additional supercells are connected into the working system. For any even N' less than N the MUSE design can be used for adaptive nulling using $3N'/2$ CORDIC cells of the type already optimized for $N = 64$. The use will not be efficient.

The approach we take is to "imbed" the length N' nulling problem into the larger length N problem. Figure B-1 illustrates how an $N' = 4$ input vector is imbedded in the middle of an $N = 10$ input vector, and how an $N' \times N'$ Cholesky matrix is therefore imbedded in the $N \times N$ Cholesky matrix developed from it. The three leading elements of the input vector are forced to zero and the last three elements may be set to any value—they are "don't care" elements indicated by d's .

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ d \\ d \\ d \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & & & & & & & & & \\ 0 & 0 & & & & & & & & \\ 0 & 0 & 0 & & & & & & & \\ 0 & 0 & 0 & l_{1,1} & & & & & & \\ 0 & 0 & 0 & l_{2,1} & l_{2,2} & & & & & \\ 0 & 0 & 0 & l_{3,1} & l_{3,2} & l_{3,3} & & & & \\ 0 & 0 & 0 & l_{4,1} & l_{4,2} & l_{4,3} & l_{4,4} & & & \\ 0 & 0 & 0 & d & d & d & d & d & & \\ 0 & 0 & 0 & d & d & d & d & d & d & \\ 0 & 0 & 0 & d & d & d & d & d & d & d \end{bmatrix}$$

Figure B-1. 4 × 4 adaptive problem imbedded in 10 × 10 framework.

If the ten-element vectors of Figure B-1 are fed into a mini-MUSE made of five supercells, the first three supercells in the five supercell chain (supercells 0, 1, and 2) will have a negligible effect on the input vectors as they are passed forward since their function is to zero out the elements which are already zero. Therefore it is easy to predict the form of the tacked-on vector that will be the input to supercell 3, namely $[x_1, x_2, x_3, x_4, d, d, d]$.

Therefore we can provide only the last two supercells (in general the last $N'/2$ supercells), and feed in the predicted vector *preceded with the proper number of blank microcycles.* Assuming that

the correct D, Q, and *ZERO* controls are provided, the two supercell MUSE will operate exactly as if it were the last two supercells of the five supercell MUSE, and the imbedded Cholesky factor will be computed. Signals can also be provided which initiate the Q-operation and the Q-op controls will be emitted from supercell 3, in its backward direction, so that the weights can be computed.

The example with two actual supercells from a chain of five supercells easily and accurately extends to the case of any smaller subset of a chain of 32 supercells. There are three points to remember:

- The actual supercells must be consecutive and must include supercell 31.
- The timing must use 67 microcycles per macrocycle and the input vector, *direction*, Q, and *ZERO* must appear as they would appear if the nonexistent supercells were present.
- The leading supercell actually present must be provided with some number of slave Q-operations to prompt it to initiate a master Q-operation at the correct time.

REFERENCES

1. C.M. Rader (private communication, 1984).

2. M. Monzingo and T. Miller, *Introduction to Adaptive Arrays*, New York: Wiley (1980), p. 94.

3. I.S. Reed, J.D. Mallett, and L.E. Brennan, "Rapid convergence in adaptive arrays," *IEEE Trans. Aerosp. Electron. Syst.* AES-10, 853 (1974).

4. C.R. Ward, P.J. Hargrave, and J.G. McWhirter, "A novel algorithm and architecture for adaptive digital beamforming," *IEEE Trans. Antennas Propag.* AP-34, No. 3, 338-346 (1986).

5. C.M. Rader and A.O. Steinhardt, "Hyperbolic householder transformations," *IEEE Trans. Acoust. Speech Signal Process.* ASSP-34(6) 1589-1602 (1986).

6. J.E. Volder, "The CORDIC trigonometric computing technique," *IEEE Trans. Electron. Comput.* EC-8(3), 330-334 (1959).

7. A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw Hill (1965), p. 65.

8. J. Raffel, A.H. Anderson, G.H. Chapman, "Laser restructurable technology and design," in Earl Swartzlander, (ed.), *Wafer Scale Integration*, Boston, MA: Kluwer Academic Publishers, (1989), pp. 319-363.

9. P.W. Wyatt and J.I. Raffel, "Restructurable VLSI – A demonstrated wafer scale technology," *Proc. of International Conference on Wafer Scale Integration,* San Francisco, CA: IEEE Computer Society Press (1989), pp. 13-20.

10. G.H. Chapman, J.M. Canter, and S.S. Cohen, "The technology of laser formed interconnections for wafer scale integration," *Proc. of International Conference on Wafer Scale Integration,* San Francisco, CA: IEEE Computer Society Press (1989) pp. 21-29.

11. R. Frankel, J.J. Hunt, M. Van Alstyne, and G. Young, "SLASH – An RVLSI CAD system," *Proc. of International Conference on Wafer Scale Integration,* San Francisco, CA: IEEE Computer Society Press (1989) pp. 31-37.

12. A.H. Anderson, R. Berger, K.H. Konkle, and F.M. Rhodes, "RVLSI applications and physical design," *Proc. of International Conference on Wafer Scale Integration,* San Francisco, CA: IEEE Computer Society Press (1989) pp. 39-45.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	18 May 1990	Technical Report

4. TITLE AND SUBTITLE
MUSE — A Systolic Array for Adaptive Nulling with 64 Degrees of Freedom, Using Given Transformations and Wafer Scale Integration

5. FUNDING NUMBERS
C — F19628-90-C-0002

6. AUTHOR(S)
C.M. Rader, D.L. Allen, D.B. Glasco, and C.E. Woodward

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Lincoln Laboratory, MIT
P.O. Box 73
Lexington, MA 02173-9108

8. PERFORMING ORGANIZATION REPORT NUMBER
TR-886

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ AF Systems Command AFSC/XTKT
Andrews AFB
Washington, DC 20334-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESD-TR-90-019

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes an architecture for a highly parallel system of computational processors specialized for real-time adaptive antenna nulling computations with many degrees of freedom, which we call MUSE, and a specific realization of MUSE for 64 degrees of freedom. Each processor uses the CORDIC algorithm and has been designed as a single integrated circuit. Ninety-six such processors working together can update the 64-element nulling weights based on 300 new observations in only 6.7 ms. This is equivalent to 2.88 Giga-ops for a conventional processor. The computations are accurate enough to support 50 dB of signal-to-noise improvement in a sidelobe canceller. The connectivity between processors is quite simple and permits MUSE to be realized on a single large wafer, using restructurable VLSI. The complete design of such a wafer is described.

14. SUBJECT TERMS			15. NUMBER OF PAGES
adaptive nulling	restructurable wafer scale integration	voltage domain computation	86
CORDIC	Givens transformations	systolic array	16. PRICE CODE
sidelobe canceller	Cholesky factor	pipeline	
		latency-controlled interleaving	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	